

LINUXTM JOURNAL

Since 1994: The Original Magazine of the Linux Community

JUNE 2014 | ISSUE 242 | www.linuxjournal.com

**E-MAIL ENCRYPTION
WITH MUTT**

**SIMPLE SCRIPTING
SKILLS IN ACTION**

**A LOOK AT HOW
URLs ARE CHANGING
WITH THE TIMES**

NETWORKING

**ANALYZE
ANDROID
TRAFFIC
WITH
WIRESHARK**

+
**HARDEN
YOUR
RHEL
SYSTEMS
WITH
ANSIBLE**

**HOW IS
MOBILE
CHANGING
THE WEB?**

**MONITOR YOUR
NETWORK
SECURITY
WITH OSSIM**

**TAKE YOUR
NETWORK
FILTERING RULES
TO A NEW LEVEL**



**WATCH:
ISSUE OVERVIEW**





2014 JUNE 13TH - 14TH

TEXAS LINUXFEST

AUSTIN, TEXAS

COME JOIN US FOR THE 5TH ANNUAL FESTIVAL.

LEARN FROM TALKS GIVEN BY INDUSTRY EXPERTS

MEET THE MANY VENDORS ON THE EXPO FLOOR

NETWORK WITH THE OPEN SOURCE COMMUNITY

REGISTER NOW AT TEXASLINUXFEST.ORG



Are you considering software-defined storage?

zStax
ZFS Unified Storage

zStax StorCore ZFS Unified Storage from Silicon Mechanics is truly software defined storage.

From modest data storage needs to a multi-tiered production storage environment, the **zStax StorCore** ZFS unified storage appliances have the right mix of performance, capacity, and reliability to fit your needs.

zStax StorCore 64



The **zStax StorCore 64** is your Tier 2 and 3 storage solution. While still leveraging all of the features inherent to the zStax platform, the StorCore 64 model offers an easily deployable and intuitively managed enterprise storage appliance. From backup and archival, to departmental file shares and streaming video, the zStax StorCore 64 has your needs covered.

zStax StorCore 104



The **zStax StorCore 104** is your system for highly available Tier 1 storage environments. Offering levels of redundancy, the StorCore 104 keeps your critical data available when competitors waiver. Finally, the StorCore 104 delivers a multi-tiered environment under one pane of management so you can eliminate the need for multiple vendors to satisfy your tiered data requirements.

CONTENTS

JUNE 2014
ISSUE 242

NETWORKING



Cover Image: © Can Stock Photo Inc. / animind

FEATURES

56 Monitoring Android Traffic with Wireshark

Get a glimpse into the Internet traffic coming from your smartphone using some simple Linux tools.

Brian Trapp

68 OSSIM: a Careful, Free and Always Available Guardian for Your Network

OSSIM, the free and open-source SIEM.

Marco Alamanni

84 Berkeley Packet Filters with Scapy (and Friends)

Decide what should come from a socket before the data even reaches your application.

Valentine Sinitsyn

INDEPTH

98 Security Hardening with Ansible

How to secure RHEL6 with Ansible.
Mark Dotson

COLUMNS

30 Reuven M. Lerner's At the Forge

URLs

36 Dave Taylor's Work the Shell

Considering Legacy
UNIX/Linux Issues

42 Kyle Rankin's Hack and /

Encrypt Your Dog (Mutt and GPG)

46 Shawn Powers' The Open-Source Classroom

Being a Hack

110 Doc Searls' EOF

In the Matrix of Mobile, Linux Is Zion

IN EVERY ISSUE

8 Current_Issue.tar.gz

10 Letters

16 UPFRONT

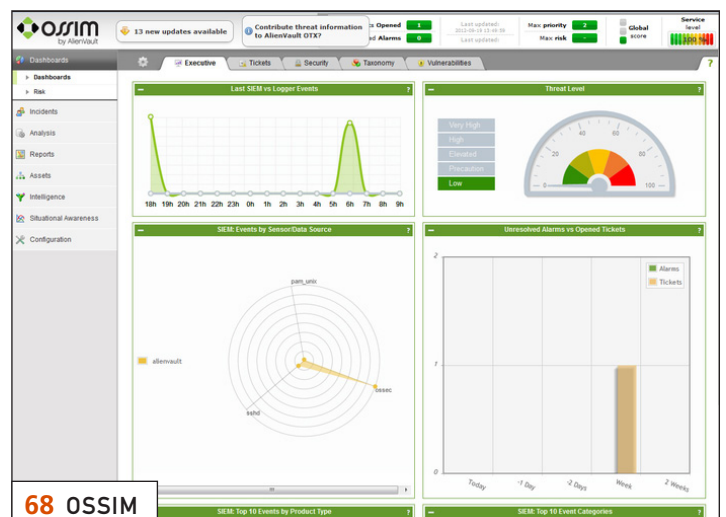
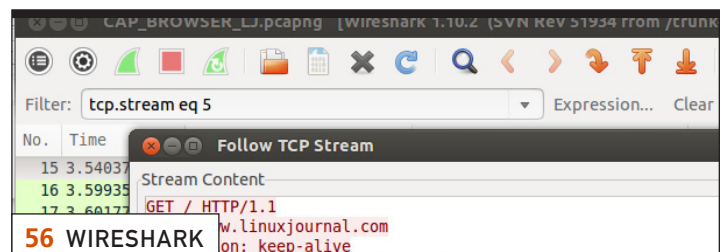
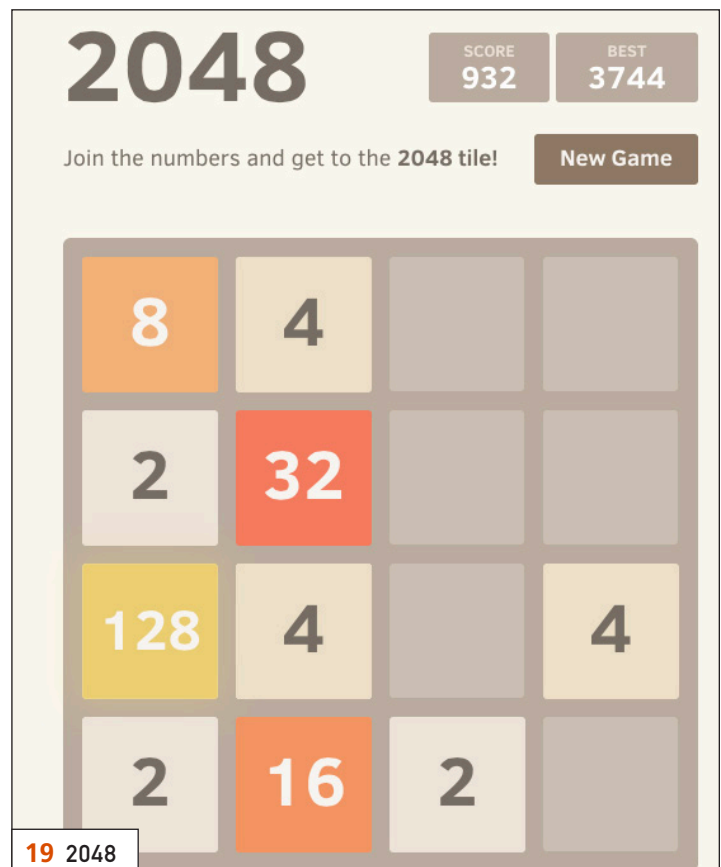
28 Editors' Choice

52 New Products

113 Advertisers Index

ON THE COVER

- E-mail Encryption with Mutt, p. 42
- Simple Scripting Skills in Action, p. 46
- A Look at How URLs Are Changing with the Times, p. 30
- Analyze Android Traffic with Wireshark, p. 56
- Monitor Your Network Security with OSSIM, p. 68
- Take Your Network Filtering Rules to a New Level, p. 84
- Harden Your RHEL Systems with Ansible, p. 98
- How Is Mobile Changing the Web?, p. 110



LINUX JOURNAL™

**Subscribe to
Linux Journal
Digital Edition
for only
\$2.45 an issue.**



ENJOY:

Timely delivery

Off-line reading

Easy navigation

Phrase search
and highlighting

Ability to save, clip
and share articles

Embedded videos

Android & iOS apps,
desktop and
e-Reader versions

SUBSCRIBE TODAY!

LINUX JOURNAL

Executive Editor Jill Franklin
jill@linuxjournal.com

Senior Editor Doc Searls
doc@linuxjournal.com

Associate Editor Shawn Powers
shawn@linuxjournal.com

Art Director Garrick Antikajian
garrick@linuxjournal.com

Products Editor James Gray
newproducts@linuxjournal.com

Editor Emeritus Don Marti
dmarti@linuxjournal.com

Technical Editor Michael Baxter
mab@cruzio.com

Senior Columnist Reuven Lerner
reuven@lerner.co.il

Security Editor Mick Bauer
mick@visi.com

Hack Editor Kyle Rankin
lj@greenfly.net

Virtual Editor Bill Childers
bill.childers@linuxjournal.com

Contributing Editors

Ibrahim Haddad • Robert Love • Zack Brown • Dave Phillips • Marco Fioretti • Ludovic Marcotte
Paul Barry • Paul McKenney • Dave Taylor • Dirk Elmendorf • Justin Ryan • Adam Monsen

Publisher Carlie Fairchild
publisher@linuxjournal.com

Director of Sales John Grogan
john@linuxjournal.com

Associate Publisher Mark Irgang
mark@linuxjournal.com

Webmistress Katherine Druckman
webmistress@linuxjournal.com

Accountant Candy Beauchamp
acct@linuxjournal.com

**Linux Journal is published by, and is a registered trade name of,
Belltown Media, Inc.**

PO Box 980985, Houston, TX 77098 USA

Editorial Advisory Panel

Brad Abram Baillio • Nick Baronian • Hari Boukis • Steve Case
Kalyana Krishna Chadalavada • Brian Conner • Caleb S. Cullen • Keir Davis
Michael Eager • Nick Faltys • Dennis Franklin Frey • Alicia Gibb
Victor Gregorio • Philip Jacob • Jay Kruizenga • David A. Lane
Steve Marquez • Dave McAllister • Carson McDonald • Craig Oda
Jeffrey D. Parent • Charnell Pugsley • Thomas Quinlan • Mike Roberts
Kristin Shoemaker • Chris D. Stark • Patrick Swartz • James Walker

Advertising

E-MAIL: ads@linuxjournal.com
URL: www.linuxjournal.com/advertising
PHONE: +1 713-344-1956 ext. 2

Subscriptions

E-MAIL: subs@linuxjournal.com
URL: www.linuxjournal.com/subscribe
MAIL: PO Box 980985, Houston, TX 77098 USA

LINUX is a registered trademark of Linus Torvalds.

LINUXTM JOURNAL DevOps

With deep focus on **Collaborative Development**, **Continuous Testing** and **Release & Deployment**, we offer here the DEFINITIVE **DevOps for Dummies**, a **mobile Application Development Primer** plus advice and help from expert sources like:

- Forrester
- Gartner
- IDC
- *Linux Journal*

Plus a host of other eBooks, videos, podcasts and more.

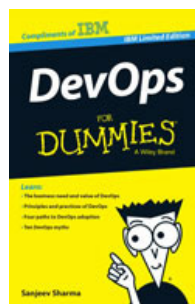
REGISTER NOW
and receive
unlimited access to
all site content and
downloads, plus
alerts when new
assets are made
available.

**Have projects in development
that need help?**

**Have a great development
operation in place that
can ALWAYS be better?**

Regardless of where you are in your DevOps process, *Linux Journal* can help!

DevOps for Dummies

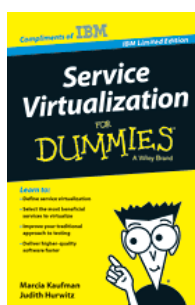


Free eBook

Today's fast-moving world makes DevOps essential for any business aspiring to be agile and lean in order to respond rapidly to changing customer and marketplace demands. This book helps you understand DevOps and how your organization can gain real business benefits from it.

You'll also discover how a holistic view of DevOps that encompasses the entire software delivery life cycle - from ideation and the conception of new business capabilities to implementation in production - can bring competitive advantage in a continuous delivery world.

Service Virtualization for Dummies Book



Free eBook

Discover service virtualization and how it fits into the big picture of software quality. In this book, Service Virtualization For Dummies, IBM Limited Edition, written by industry analysts Marcia Kaufman and Judith Hurwitz, learn how to deliver higher quality software by increasing the efficiency and effectiveness of your testing processes while reducing testing downtime and testing cost.

REGISTER NOW
<http://devops.linuxjournal.com>



SHAWN POWERS

(Net)working the Room

I tend to be a fairly funny guy. Well, at least I think I'm funny. My kids might disagree. The thing is, it's hard to find a group of people to understand obscure networking jokes. At a non-tech conference, for example, if I say to the person next to me, "Jeez, that speaker must have delivered his presentation with UDP packets, because he never stopped to see if any of us were getting what he was talking about!"—exactly zero people laugh. In fact, I usually get really weird looks. At a Linux conference, however, the same comment usually gets an eye-roll. (As a father of teenagers, I consider an eye-roll the equivalent to amusement.) That's why I love *Linux Journal* so much. This month, we're talking about Networking, and everyone in our little "room" understands what we're talking about! So let's peel this issue apart one OSI layer at a time.

Reuven M. Lerner starts us out with

URLs. That ubiquitous string of text that takes you to a location (usually a Web site) is something we often take for granted. As the Internet matures, however, an understanding of how URLs work is vital. Reuven teaches us everything from protocol designations to URL fragments. If you've ever wondered about those seemingly out of place # characters in a URL, you'll want to read his column. Dave Taylor follows with a great look at the evolution of scripting. Just like we no longer have to hand-crank our car engine to get it running (mine doesn't even have a key anymore, just a button), shell scripting has changed through the years. Supporting legacy systems (or legacy code) is a problem we all need to deal with, as Dave shows us with one of his real-world experiences.

Kyle Rankin continues his theme this month and teaches how to encrypt our e-mail—specifically text-based Mutt e-mail. Kyle remains true to his command-line preferences, and rather than switch to a GUI e-mail app, he



VIDEO:
Shawn Powers runs
through the latest issue.

describes how to use GPG with Mutt. If you're a Mutt user like Kyle, or just want to learn about implementing GPG, don't miss his column. I follow Kyle with a continuation on last month's scripting basics article. Rather than leaving you with a simple set of tools, I tried to come up with some examples of using those tools in real-world situations. My scripts are basic and my techniques are simple, but that's the point. Don't be intimidated by the command line. It's powerful and not terribly difficult to master.

Sniffing network traffic is often a critical part of diagnosing a problem or detecting a potential one. Brian Trapp explains how to sniff the packets of our smartphones. That seems like a simple enough task, until you realize capturing traffic from a wireless device to the network can be challenging. (Using Firesheep will garner you tons of Web information, but if you want every packet, that gets a little rougher.) Brian shows how to capture the traffic, and then how to dissect it with Wireshark. Marco Alamanni follows Brian with an in-depth article on OSSIM, a server-based program for detecting problems on your local network. With its Web interface and powerful collection of snooping tools, OSSIM can be incredibly useful for early detection of problems or threats.

Any network user or administrator is

familiar with firewalling tools. Most of us, however, aren't nearly as familiar with Berkeley Packet Filters. Valentine Sinitsyn walks through using BPF to do some very low-level traffic filtering. This powerful system can add an incredible set of tools to your network filtering needs. And finally, Mark Dotson gives us an in-depth tutorial on Ansible. Managing multiple systems is becoming more and more complex, but thankfully with tools like Ansible, we can deploy and configure countless machines quickly and, most importantly, securely.

Like every issue of *Linux Journal*, this one is chock full of tech tips, product announcements and recommendations. The networking issue touches on so many disciplines and interest areas in the Linux community, that it's always one of our favorites. The large majority of folks still won't understand our networking jokes, but that's okay, they can sit around as bored as a teenager in a Faraday cage while we all enjoy this issue. (Thank you, thank you, I'll be here all night....)■

Shawn Powers is the Associate Editor for *Linux Journal*. He's also the Gadget Guy for LinuxJournal.com, and he has an interesting collection of vintage Garfield coffee mugs. Don't let his silly hairdo fool you, he's a pretty ordinary guy and can be reached via e-mail at shawn@linuxjournal.com. Or, swing by the #linuxjournal IRC channel on Freenode.net.

letters



Software Usability

Jim Hall's article, "It's about the User: Applying Usability in Open-Source Software" in the December 2013 issue, was right on the mark and also applies to custom in-house software used by large companies. I was going to cite a number of examples but decided to keep this "short and sweet", or should I say "short and sour". There are plenty of examples, but today, I'm picking on just one, Kompozer.

I used Kompozer on an XP machine, and it did what I wanted to do. So, when I said good-bye to Windows, I wanted Kompozer or an equivalent program on my new Ubuntu 13.1 box.

Try installing Kompozer—I challenge you! The Internet has countless pages of questions and answers representing thousands of wasted hours by people around the globe. After following the step-by-step instructions, using copy and paste so I didn't make a mistake, I still don't have it running.

This is the very sort of thing that has been holding Linux back from becoming mainstream! It kept me and others that I know from taking the plunge years ago. All it would take is a few minutes from *one* person who knows what they're doing to package it properly and save thousands of hours of frustration for Linux users everywhere!

—Graig

Jim Hall replies: *Thanks for your e-mail! You are correct that usability applies both to commercial/proprietary software and to free/open-source software. When you are intimately involved with the development of software, you become so familiar with it and what it does that it's easy to forget that ordinary people with average knowledge need to use the software. Usability means focusing on the users and what they need to do with the software. It's dangerous*

to assume that everyone will use the software the way you want them to.

Harden Your SSH

Regarding Federico Kereki's article "More Secure SSH" in the January 2014 issue, using PAM, how can you lock a user to a directory? Would it be better to use ChrootDirectory? I would think PAM would be better, but I am lost at the moment. I have never configured PAM, but I am interested in it because of this article.

—Tony Catalfamo

Federico Kereki replies: *I haven't actually used this, but there's a PAM module you could try called pam_chroot. However, I'd also point out that according to some accounts, there might be security problems with it, and jk_chrootsh should be preferred. As I said, I haven't tried this out, but I'll give these two options a whirl; good question, Tony!*

Another Silo on the Security Topic

Some years ago, I bought and installed a couple door locks with the Z-Wave interface feature, thinking that someday I would integrate them into what we call The Internet of Things. Today, I find that my *only* option

with the manufacturer Schlage is to incorporate its own Web site into the path from lock to PC or smartphone.

Although the salesman offers glowing prospects, he has no way to deal with the contingencies I presented to him that are important failures to avert per my values. Doc Searls and others have been writing about silos for quite some time. Short of reverse-engineering, I see no way to avoid yet another one in my personal life.

This touches many topics actually—for example, security, embedded systems, service interruption, silos and the "biggie", Internet of Things. I have regaled in some of Docs' successes in beating the lock-in of vendors. While my search for an alternative door-lock vendor is not done, I wonder if a "Silo Beaters Registry" might make good reading? Like open source, it would carry those who are receptive to providing open products of non-software nature—like door locks.

In any case, should Doc or others write more on the silo topic, I present this example to them through you for their use.

—Skip

Doc Searls replies: *Thanks, Skip. I feel your pain—and I like the idea of a “Silo-Beater’s Registry”.*

I also would love to hear from other readers about what the qualifications would be. Meanwhile, I’ll collect my own thoughts on the matter. Thanks again.

Linux Backup: the New Bareos vs. Bacula

It seems there is a new fork of Bacula that deserves some attention, and I’d love to see *LJ* do a review if you have room in your queue (<http://www.bareos.org>).

Basically, a couple main contributors have forked from just before where Bacula Enterprise code went proprietary.

It isn’t currently available via any of the distros other than Gentoo, but the authors do have packaged binaries for the majors.

It’s well worth a look, and I’d love to hear what you think!

—**Erich Newell**

Very cool! I’ll try to check it out, but at the very least, it will appear here for others to check out as well!—Shawn Powers

New Android App

Recently, a new *Linux Journal* app was automatically downloaded to my Android phone replacing the existing *Linux Journal* app. The UI has been improved. I really appreciate this. But there are a few problems:

1) The new app will start, but then refuses to open a previously downloaded issue, unless it has Internet access. I don’t think going off-line while reading prevents further reading of the same issue.

2) The new app crashes just after start once in a while.

With the “old” version of the app, I believe I could open downloaded issues without having Internet access. It is quite annoying at the moment, because I’m abroad and thus only connected to the Internet sporadically.

Will you consider bringing back the functionality to start reading while off-line?

—**Bo Romer**

The app development isn’t done in-house, but I’ll be sure to get your feedback to the right folks. Thank you for the input; the goal is, of course, to have the most

usable, convenient app we can provide.—Shawn Powers

Linux Journal Content

I am writing to your team to request that you please provide more articles regarding system administration of Linux in future issues—for example, useful commands, explaining the `inet.d` or `.rc` dirs, troubleshooting tips, slow performance evaluations and best practices configurations. I could go on, but I think you get the point. I enjoy reading these topics, as it keeps me abreast of new things

and refreshes my memory on others, and I haven't seen this content in your past issues. Feel free to reach out to me if you like.

—Ron

Sysadmin topics are usually pretty popular, and it's one of the reasons we added my "Open-Source Classroom" column. We will continue to have issues that focus specifically on system administration, but I'll try to work some of the topics you mention into my column if nothing else. Thanks for the great ideas!—Shawn Powers

Powerful: Rhino



Rhino M4800/M6800

- Dell Precision M6800 w/ Core i7 Quad (8 core)
- 15.6"-17.3" QHD+ LED w/ X@3200x1800
- NVidia Quadro K5100M
- 750 GB - 1 TB hard drive
- Up to 32 GB RAM (1866 MHz)
- DVD±RW or Blu-ray
- 802.11a/b/g/n
- Starts at \$1375
- E6230, E6330, E6440, E6540 also available

- High performance NVidia 3-D on an QHD+ RGB/LED
- High performance Core i7 Quad CPUs, 32 GB RAM
- Ultimate configurability — choose your laptop's features
- One year Linux tech support — phone and email
- Three year manufacturer's on-site warranty
- Choice of pre-installed Linux distribution:



Tablet: Raven



Raven X240

- ThinkPad X240 by Lenovo
- 12.5" FHD LED w/ X@1920x1080
- 2.6-2.9 GHz Core i7
- Up to 16 GB RAM
- 180-256 GB SSD
- Starts at \$1910
- W540, T440, T540 also available

Rugged: Tarantula



Tarantula CF-31

- Panasonic Toughbook CF-31
- Fully rugged MIL-SPEC-810G tested: drops, dust, moisture & more
- 13.1" XGA TouchScreen
- 2.4-2.8 GHz Core i5
- Up to 16 GB RAM
- 320-750 GB hard drive / 512 GB SSD
- CF-19, CF-52, CF-H2, FZ-G1 available

EmperorLinux
...where Linux & laptops converge

www.EmperorLinux.com
1-888-651-6686



Wrong Info in “Understanding Caching” Article

I’m writing about the article “Understanding Caching” published in January 2004, but still available on-line: <http://www.linuxjournal.com/article/7105>.

The author begins his article by stating processors have always been faster than memory. It’s just plain wrong; early CPUs were slower than memory. Actually memory (SDRAM) density has greatly improved (as much as CPUs), not latencies as much.

Proofs:

1) Princeton Physics Laboratory:
<http://w3.pppl.gov/~hammett/comp/bench/bandwidth.html>.

Figure 2 in this article: http://www.dba-oracle.com/oracle_tips_hardware_oracle_performance.htm.
—Eric

|||||

WRITE LJ A LETTER

We love hearing from our readers. Please send us your comments and feedback via <http://www.linuxjournal.com/contact>.

PHOTO OF THE MONTH

Remember, send your Linux-related photos to ljeditor@linuxjournal.com!

LINUX JOURNAL

At Your Service

SUBSCRIPTIONS: *Linux Journal* is available in a variety of digital formats, including PDF, .epub, .mobi and an on-line digital edition, as well as apps for iOS and Android devices. Renewing your subscription, changing your e-mail address for issue delivery, paying your invoice, viewing your account details or other subscription inquiries can be done instantly on-line: <http://www.linuxjournal.com/subs>. E-mail us at subs@linuxjournal.com or reach us via postal mail at *Linux Journal*, PO Box 980985, Houston, TX 77098 USA. Please remember to include your complete name and address when contacting us.

ACCESSING THE DIGITAL ARCHIVE: Your monthly download notifications will have links to the various formats and to the digital archive. To access the digital archive at any time, log in at <http://www.linuxjournal.com/digital>.

LETTERS TO THE EDITOR: We welcome your letters and encourage you to submit them at <http://www.linuxjournal.com/contact> or mail them to *Linux Journal*, PO Box 980985, Houston, TX 77098 USA. Letters may be edited for space and clarity.

WRITING FOR US: We always are looking for contributed articles, tutorials and real-world stories for the magazine. An author’s guide, a list of topics and due dates can be found on-line: <http://www.linuxjournal.com/author>.

FREE e-NEWSLETTERS: *Linux Journal* editors publish newsletters on both a weekly and monthly basis. Receive late-breaking news, technical tips and tricks, an inside look at upcoming issues and links to in-depth stories featured on <http://www.linuxjournal.com>. Subscribe for free today: <http://www.linuxjournal.com/enewsletters>.

ADVERTISING: *Linux Journal* is a great resource for readers and advertisers alike. Request a media kit, view our current editorial calendar and advertising due dates, or learn more about other advertising and marketing opportunities by visiting us on-line: <http://www.linuxjournal.com/advertising>. Contact us directly for further information: ads@linuxjournal.com or +1 713-344-1956 ext. 2.



Instant Access to Premium Online Drupal Training

- ✓ *Instant access to hundreds of hours of Drupal training with new videos added every week!*
- ✓ *Learn from industry experts with real world experience building high profile sites*
- ✓ *Learn on the go wherever you are with apps for iOS, Android & Roku*
- ✓ *We also offer group accounts. Give your whole team access at a discounted rate!*

Learn about our latest video releases and offers first by following us on Facebook and Twitter (@drupalizeme)!

Go to <http://drupalize.me> and get Drupalized today!



diff -u

WHAT'S NEW IN KERNEL DEVELOPMENT

David Herrmann wanted to disable the **virtual terminal subsystem** in order to save space on a kernel that didn't need a VT. But, he still wanted to see kernel oops output for debugging purposes. The problem was that only the VT subsystem would display oops output—and he'd just disabled it.

No problem. David posted a patch to implement **DRM-log**, a separate console device that used the direct rendering manager and that could receive kernel oops output.

Over the course of a discussion about the patch, **Alan Cox** mentioned that there didn't seem to be anything particularly DRM-specific in David's code. It easily could exist at a yet more generic layer of the kernel. And although David agreed with this, he said the DRM folks were more amenable to taking his patch and that "I've spent enough time trying to get the attention of core maintainers for simple fixes, I

really don't want to waste my time pinging on feature-patches every 5 days to get any attention. If someone outside of DRM wants to use it, I'd be happy to discuss any code-sharing. Until then, I'd like to keep it here as people are willing to take it through their tree."

That's a fairly surprising statement—a bit of an indictment of existing kernel patch submission processes. There was no further discussion on that particular point, but I would imagine it got some folks thinking.

The rest of the current thread focused on some technical details about oops output, especially font size. David's code displayed oops output pixel by pixel, essentially defining its own font. But for extremely high-resolution monitors, such as **Apple's Retina display**, as **Bruno Prémont** pointed out, this could result in the oops output being too small for the user to see.

David's answer to this was to implement integer scaling. His font could be any integer multiple larger

than the default. This seemed fine to Bruno.

Eugene Shatokhin posted some code to make use of **Google's ThreadSanitizer** tool (<https://code.google.com/p/thread-sanitizer>). ThreadSanitizer detects a particular type of race condition that occurs when one thread tries to write to a variable while another thread either tries to read from or write to the same variable.

Eugene called his own code **Kernel Strider** (<https://code.google.com/p/kernel-strider>). It collected statistics on memory accesses, function calls and other things, and sent them along to be analyzed by Thread Sanitizer. Eugene also posted a link to a page describing several race conditions that Kernel Strider had uncovered in the 3.10.x kernel series.

Waiman Long posted some code implementing **qspinlock**, a new type of spinlock that seemed to improve speed on very large multiprocessor systems. The idea behind the speed improvement was that a CPU would disable preemption when spinning for a lock, so it would save the time that might otherwise have been used

migrating the looping thread to other CPUs.

The big problem with that kind of improvement is that it's very context-dependent. What's faster to one user may be slower to another, depending on one's particular usual load. Traditionally, there has been no clean way to resolve that issue, because there really is not any "standard" load under which to test the kernel. The developers just have to wing it.

But, they wing it pretty good, and ultimately things like new spinlock implementations do get sufficient testing to determine whether they'd be a real improvement. The problem with Waiman's situation, as he said on the list, is that the qspinlock implementation is actually slower than the existing alternatives on systems with only a few CPUs—in other words, for anyone using Linux at home.

However, as **George Spelvin** pointed out, the most common case is when a spinlock doesn't spin even once, but simply requests and receives the resource in question. And in that case, qspinlocks seem to be just as fast as the alternatives.

To qspinlock or not to qspinlock—

Rik van Riel knew his answer and sent out his “Signed-Off-By” for Waiman’s patch. Its merits undoubtedly will continue to be tested and debated. But there are many, many locking implementations in the kernel. I’m sure this one will be used somewhere, even if it’s not used everywhere.

Yuyang Du recently suggested separating the **Linux scheduler** into two independent subsystems: one that performed load balancing between CPUs and the other that actually scheduled processes on each single CPU.

The goal was lofty. With the scheduler performing both tasks, it becomes terribly complex. By splitting it into these two halves, it might become possible to write alternative systems for one or the other half, without messing up the other.

But in fact, no. There was almost universal rejection of the idea. *Peter Zijlstra* said, “That’s not ever going to happen.” **Morten Rasmussen** said the two halves couldn’t be separated the way Yuyang wanted—they were inextricably bound together.

You never know though. Once upon a time, someone said Linux never would support any architecture other than i386. Now it runs on anything that contains silicon, and there’s undoubtedly an effort underway to port it to the human brain. Maybe the schedule can be split into two independent halves as well.—**ZACK BROWN**

They Said It

**Work and acquire,
and thou hast
chained the wheel
of Chance.**

—**Ralph Waldo
Emerson**

**Our lives begin to
end the day we
become silent about
things that matter.**

—**Martin Luther
King Jr.**

**Millions long for
immortality who
don’t know what to
do with themselves
on a rainy Sunday
afternoon.**

—**Susan Ertz**

**People laugh at me
because I use big
words. But if you
have big ideas you
have to use big
words to express
them, haven’t you?**

—**L. M. Montgomery**

**Our patience will
achieve more than
our force.**

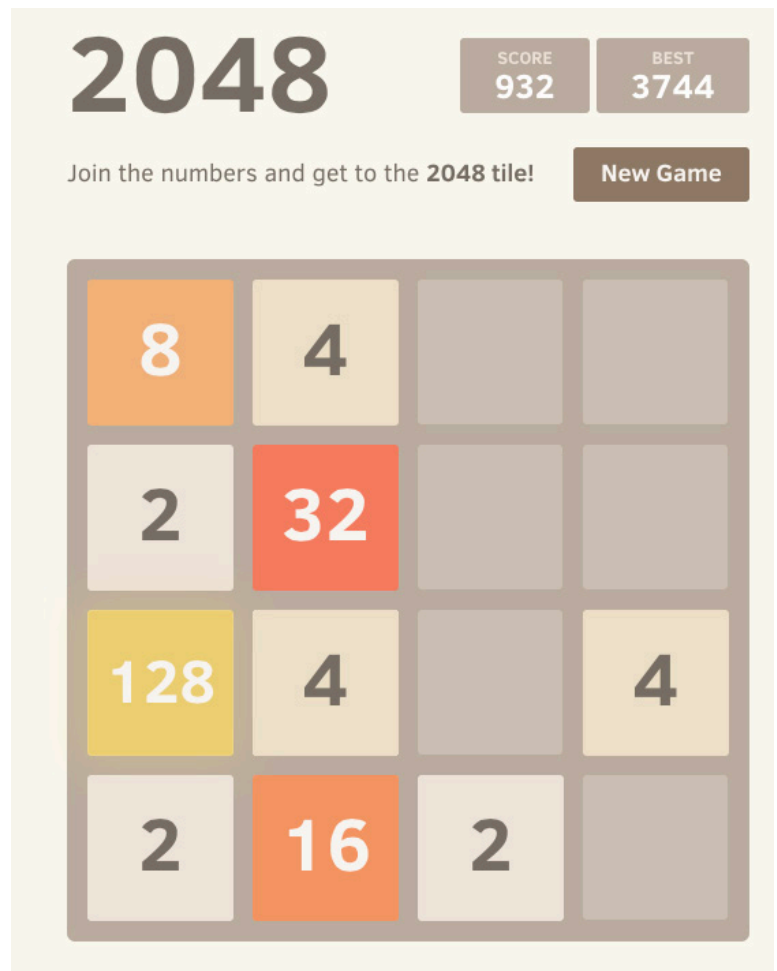
—**Edmund Burke**

Android Candy: I'm Sorry 2048 Times

It seems like every day there's a new mobile game that takes the world by storm. Whether it's *Flappy Bird* or *Candy Crush*, there's something about simple games that appeals to our need for quick, instant gratification.

I don't normally recommend games, unless they're particularly nostalgic or something, but this month I have to mention *2048*. Maybe it's the math nerd in me that loves powers of 2, or maybe it's that this game is just the right amount of challenging and infuriating. Whatever the secret recipe for a great mobile game is, *2048* has discovered it.

The basic premise is you keep combining similarly numbered blocks to get higher and higher numbered blocks. To win the game, you get the coveted 2048 block. I know our own *Linux Journal* bookkeeper has gotten further than that, however, and has scored at least 4096, with rumors of getting as high as 8192. Do you like math? Do you hate sleep? This game might be just for you. And like the title

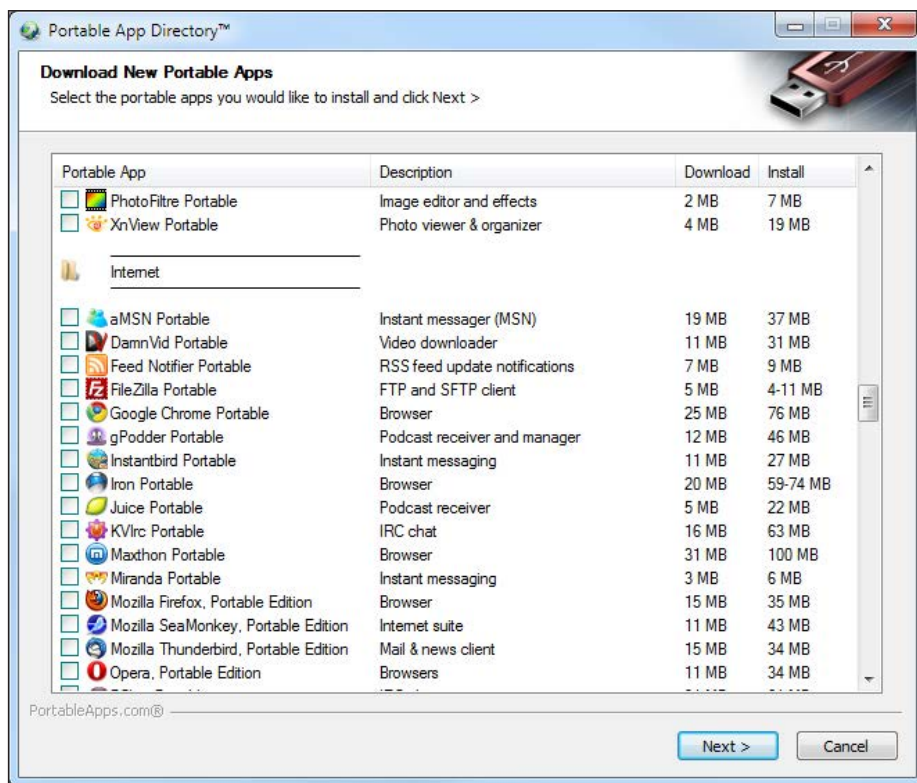


says, I'm so, so sorry. Get your copy today—just search for “2048” at the Google Play store. (There are several similar games, I don't want to favor one over the other.)

If you don't want to play on your phone, you can play on-line at <http://gabrielecirulli.github.io/2048>.

—SHAWN POWERS

Non-Linux FOSS: My Portable Windows Lab



when I get back home, because Windows can be a breeding ground for nasty infections. In order to build a USB device quickly that I can use to help my Windows friends, I like to use the awesome open-source program at <http://portableapps.com>.

The downloadable program provides a sort of “app store” for downloading individual portable apps. It makes sure all of your apps

Portable apps aren’t anything new. There are variations of “single executable apps” for most platforms, and some people swear by keeping their own applications with them for use when away from home. I don’t usually do that, as most of what I do is on-line, but there is one exception: security.

When I’m asked to help a Windows user figure out what is wrong with his or her computer, I generally take a USB drive and nothing else. I also usually run dd on that Flash drive

are up to date, and it’s a great way to browse the different categories and look for apps that might be useful. Granted, many of the portable apps themselves aren’t open source, but the program that manages them for you is. If you ever need to help friends or acquaintances with their infected systems, a USB drive prepped with the Windows-based portableapps.com application is a great way to start.

—SHAWN POWERS

rc.local, Cron Style

Occasionally as seasoned Linux users, we run across simple things we never knew existed—and are amazed. Whether it's tab autocompletion, `sudo !!` for when you forgot to type `sudo` or even recursive file listing with `ls`, the smallest tricks can be so incredibly useful. Not long ago, I had one of those moments.

Most people know `rc.local` is the file where you put commands you want to have start on system boot. Sometimes the `rc.local` script is disabled, however, and it doesn't work. It also can be difficult to remember the syntax for starting a particular program as a specific user. Plus, having a long list of programs in `rc.local` can just become ugly. Little did I know, cron supports not only periodic execution of commands, but it also can start programs when the system starts as well!

A normal crontab entry looks like this:

```
* * * * * /usr/bin/command
```

That runs the command every minute. There are countless variations to get very specific intervals, but until recently, I didn't know there were options to the five fields. The following is a crontab entry that runs



a command every hour on the hour:

```
@hourly /usr/bin/command
```

And, there are many more: `@annually`, `@monthly`, `@daily`, `@midnight` and most interesting for this article, `@reboot`. If you have a crontab entry like this:

```
@reboot /usr/bin/command
```

it will execute when the system starts up, with the ownership and permission of the person owning the crontab! I researched a lot to make sure it wasn't just on reboot, but also on a cold boot—and yes, the `@reboot` terminology just means it runs once when the system first boots. I've been using this as a quick hack to start programs, and it works amazingly well.

I know 99.9% of you already knew this juicy bit of info, but for that .1% who have been living in the dark like me, I present you with a sharp new arrow for your system administrator quiver. It's a very simple trick, but all the best ones are!—**SHAWN POWERS**

Scientific Graphing in Python

In my last few articles, I looked at several different Python modules that are useful for doing computations. But, what tools are available to help you analyze the results from those computations? Although you could do some statistical analysis, sometimes the best tool is a graphical representation of the results. The human mind is extremely good at spotting patterns and seeing trends in visual information. To this end, the standard Python module for this type of work is matplotlib (<http://matplotlib.org>). With matplotlib, you can create complex graphics of your data to help you discover relations.

You always can install matplotlib from source; however, it's easier to install it from your distribution's package manager. For example, in Debian-based distributions, you would install it with this:

```
sudo apt-get install python-matplotlib
```

The python-matplotlib-doc package also includes extra documentation for matplotlib.

Like other large Python modules,

matplotlib is broken down into several sub-modules. Let's start with pyplot. This sub-module contains most of the functions you will want to use to graph your data. Because of the long names involved, you likely will want to import it as something shorter. In the following examples, I'm using:

```
import matplotlib.pyplot as plt
```

The underlying design of matplotlib is modeled on the graphics module for the R statistical software package. The graphical functions are broken down into two broad categories: high-level functions and low-level functions. These functions don't work directly with your screen. All of the graphic generation and manipulation happens via an abstract graphical display device. This means the functions behave the same way, and all of the display details are handled by the graphics device. These graphics devices may represent display screens, printers or even file storage formats. The general work flow is to do all of your drawing in memory on the abstract graphics device. You

then push the final image out to the physical device in one go.

The simplest example is to plot a series of numbers stored as a list. The code looks like this:

```
plt.plot([1,2,3,4,3,2,1]) plt.show()
```

The first command plots the data

stored in the given list in a regular scatterplot. If you have a single list of values, they are assumed to be the y-values, with the list index giving the x-values. Because you did not set up a specific graphics device, matplotlib assumes a default device mapped to whatever physical display you are using. After executing the first

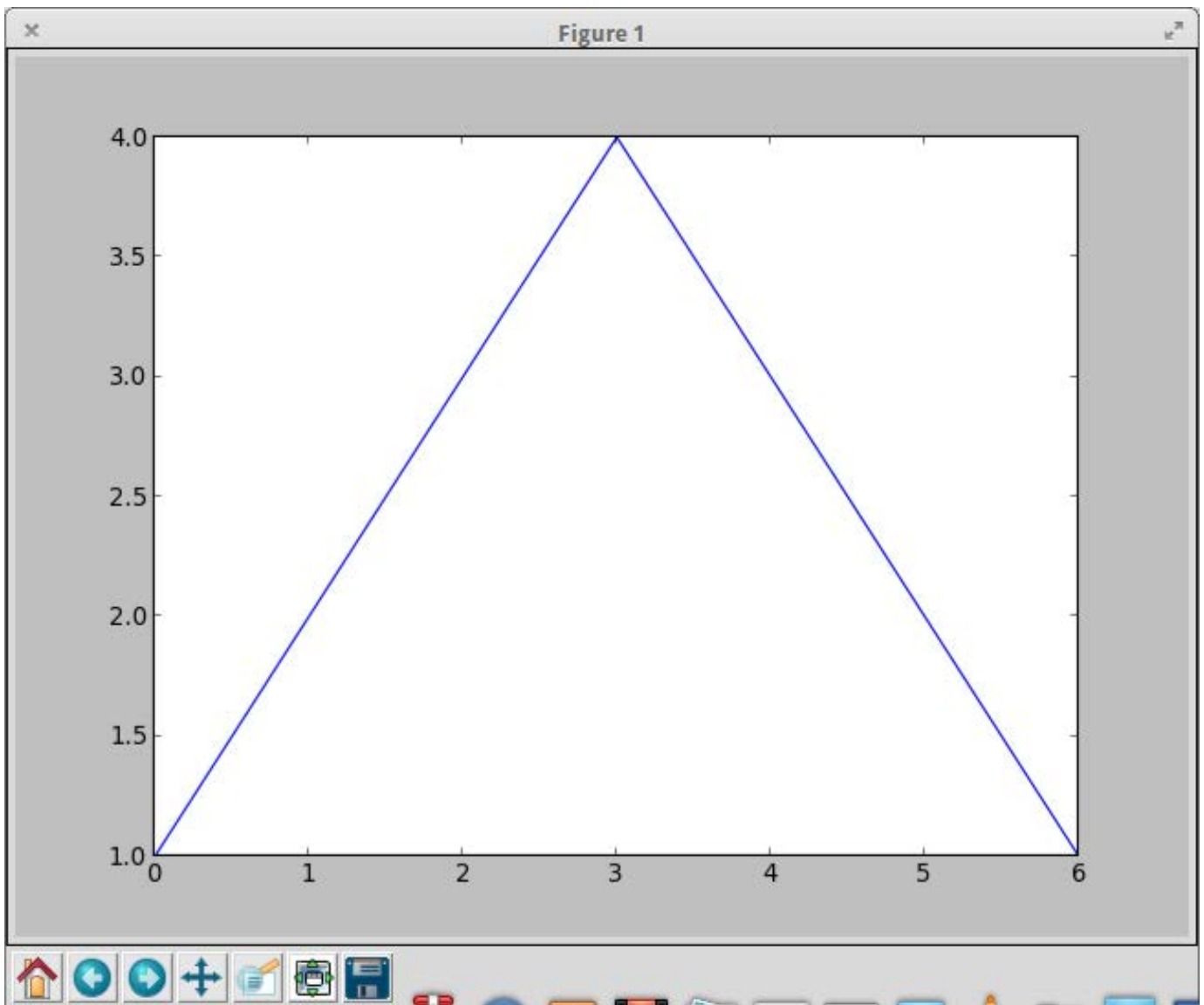


Figure 1. A basic scatterplot window includes controls on the bottom of the pane.

[UPFRONT]

line, you won't see anything on your display. To see something, you need to execute the second `show()` command. This pushes the graphics data out to the physical display (Figure 1). You should notice that there are several control buttons along the bottom of the window, allowing you to do things like save the image to a file. You

also will notice that the graph you generated is rather plain. You can add labels with these commands:

```
plt.xlabel('Index')  
plt.ylabel('Power Level')
```

You then get a graph with a bit more context (Figure 2). You can add a title for

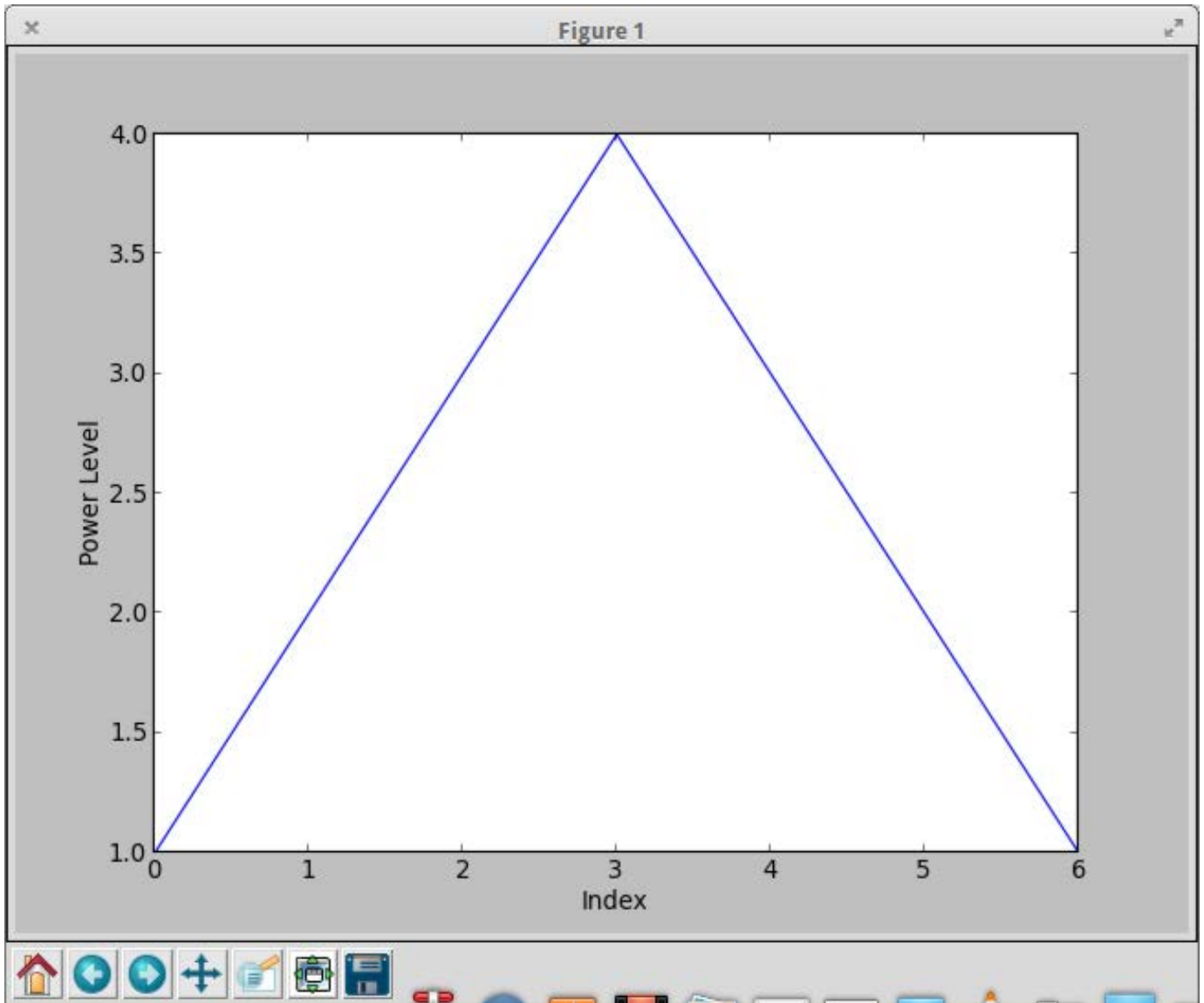


Figure 2. You can add labels with the `xlabel` and `ylabel` functions.

your plot with the `title()` command, and the `plot` command is even more versatile than that. You can change the plot graphic being used, along with the color. For example, you can make green triangles by adding `g^` or blue circles with `bo`. If you want more than one plot in a single window, you simply add them as extra options to `plot()`. So,

you could plot squares and cubes on the same plot with something like this:

```
t = [1.0,2.0,3.0,4.0]
plt.plot(t,[1.0,4.0,9.0,16.0], 'bo', t,[1.0,8.0,27.0,64.0], 'sr')
plt.show()
```

Now you should see both sets of data in the new plot window (Figure

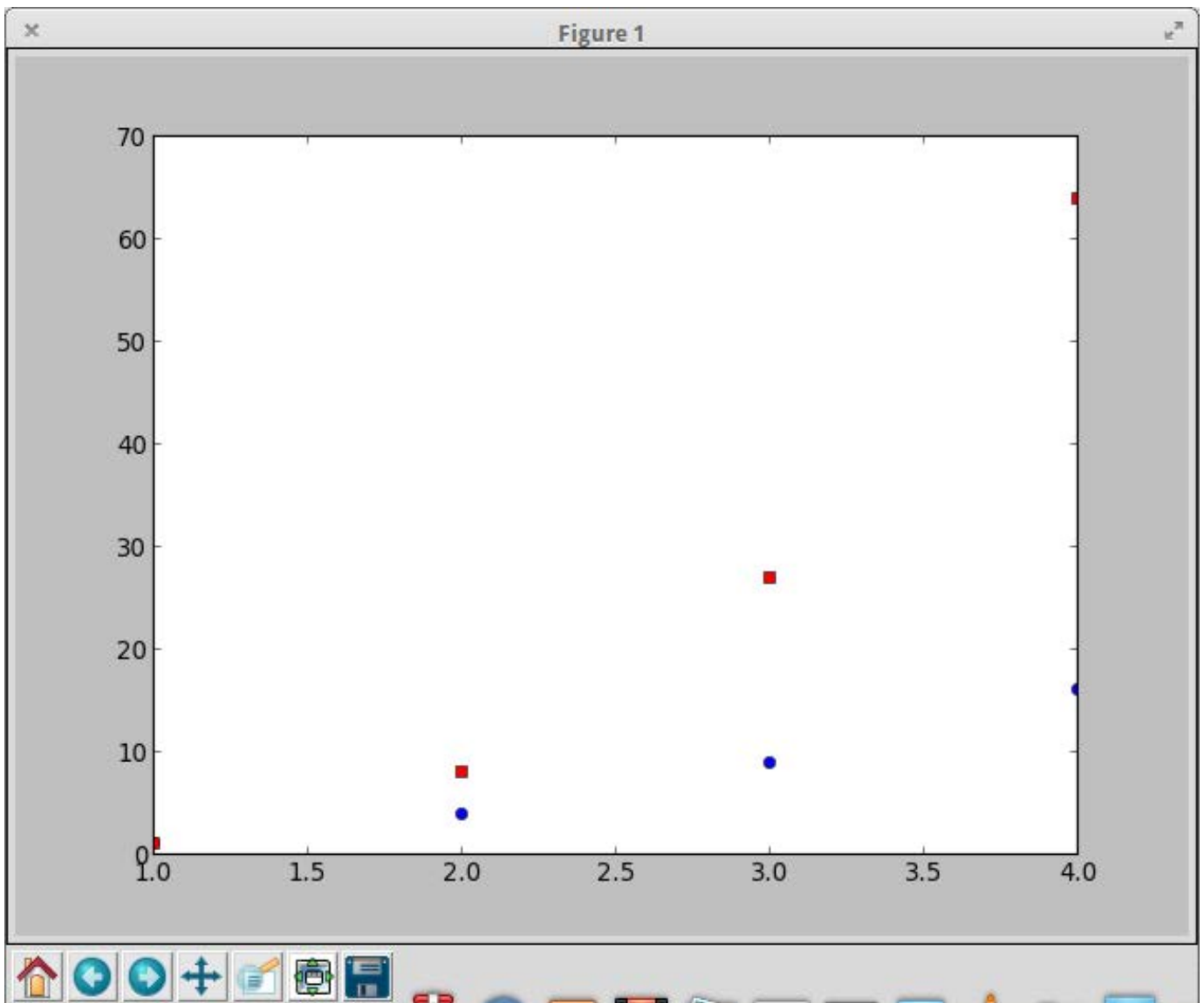


Figure 3. You can draw multiple plots with a single command.

3). If you import the numpy module and use arrays, you can simplify the plot command to:

```
plt.plot(t,t**2,'bo',t,t**3,'sr')
```

What if you want to add some more information to your plot, maybe a text box? You can do that with the `text()` command, and you can set the location for your text box, along with its contents. For example, you could use:

```
plt.text(3,3,'This is my plot')
```

This will put a text area at $x=3$, $y=3$.

A specialized form of text box is an annotation. This is a text box linked to a specific point of data. You can define the location of the text box with the `xytext` parameter and the location of the point of interest with the `xy` parameter. You even can set the details of the arrow connecting the two with the `arrowprops` parameter. An example may look like this:

```
plt.annotate('Max value', xy=(2, 1), xytext=(3, 1.5),  
            ➡arrowprops=dict(facecolor='black', shrink=0.05),)
```

Several other high-level plotting commands are available. The `bar()` command lets you draw a barplot of your data. You can change the width, height and colors with various

input parameters. You even can add in error bars with the `xerr` and `yerr` parameters. Similarly, you can draw a horizontal bar plot with the `barh()` command. Or, you can draw box and whisker plots with the `boxplot()` command. You can create plain contour plots with the `contour()` command. If you want filled-in contour plots, use `contourf()`. The `hist()` command will draw a histogram, with options to control items like the bin size. There is even a command called `xkcd()` that sets a number of parameters so all of the subsequent drawings will be in the same style as the xkcd comics.

Sometimes, you may want to be able to interact with your graphics. matplotlib needs to interact with several different toolkits, like GTK or Qt. But, you don't want to have to write code for every possible toolkit. The pyplot sub-module includes the ability to add event handlers in a GUI-agnostic way. The `FigureCanvasBase` class contains a function called `mpl_connect()`, which you can use to connect some callback function to an event. For example, say you have a function called `onClick()`. You can attach it to the button press event with this command:

```
fig = plt.figure()  
...  
cid = fig.canvas.mpl_connect('button_press_event', onClick)
```

Now when your plot gets a mouse click, it will fire your callback function. It returns a connection ID, stored in the variable `cid` in this example, that you can use to work with this callback function. When you are done with the interaction, disconnect the callback function with:

```
fig.canvas.mpl_disconnect(cid)
```

If you just need to do basic interaction, you can use the `ginput()` command. It will listen for a set amount of time and return a list of all of the clicks that happen on your plot. You then can process those clicks and do some kind of interactive work.

The last thing I want to cover here is animation. `matplotlib` includes a sub-module called `animation` that provides all the functionality that you need to generate MPEG videos of your data. These movies can be made up of frames of various file formats, including PNG, JPEG or TIFF. There is a base class, called `Animation`, that you can subclass and add extra functionality. If you aren't interested in doing too much work, there are included subclasses. One of them, `FuncAnimation`, can generate an animation by repeatedly applying a given function and generating the frames of your animation. Several

other low-level functions are available to control creating, encoding and writing movie files. You should have all the control you require to generate any movie files you may need.

Now that you have `matplotlib` under your belt, you can generate some really stunning visuals for your latest paper. Also, you will be able to find new and interesting relationships by graphing them. So, go check your data and see what might be hidden there.—JOEY BERNARD

LINUX JOURNAL ARCHIVE DVD



NOW AVAILABLE
www.linuxjournal.com/dvd

Sicker Than Sickbeard?



The screenshot shows the NZBDrone web interface. At the top is a navigation bar with icons for Series, Calendar, History, Wanted, Settings, System, and Donate. Below this is a search bar. The main content area has a toolbar with buttons for '+ Add Series', 'Season Pass', 'Series Editor', 'RSS Sync', and 'Update Library'. On the right, there are 'Sort' and view toggle buttons. The central part of the interface is a table listing TV series.

Title	Seasons	Quality	Network	Next Airing	Episodes	
▶ 24	9	SD	FOX	Today	192 / 192	⚙️ ✕
■ 3rd Rock from the Sun	6	SD	NBC		139 / 139	⚙️ ✕
▶ Under the Dome	2	HD-720p	CBS	2 months	13 / 13	⚙️ ✕
■ Alias	5	SD	ABC		105 / 105	⚙️ ✕
■ Almost Human	1	HD-720p	FOX		13 / 13	⚙️ ✕
■ Alphas	2	SD	Syfy		24 / 24	⚙️ ✕
■ Andromeda	5	SD	SciFi		110 / 110	⚙️ ✕
▶ Bones	9	HD-720p	FOX	7 days	188 / 188	⚙️ ✕
▶ Castle (2009)	6	HD-720p	ABC	Today	126 / 126	⚙️ ✕
▶ The Middle	5	HD-720p	ABC	Wednesday	116 / 116	⚙️ ✕
▶ The Millers	2	HD-720p	CBS	Thursday	21 / 21	⚙️ ✕
■ Animal Practice	1	HD-720p	NBC		9 / 9	⚙️ ✕

When I wrote about Usenet and Sickbeard a while back, I got many e-mails that I had broken the first rule of Usenet: don't talk about Usenet. I'm a sucker for freedom though, and I can't help but share when cool programs are available. This month, I switched from Sickbeard to NZBDrone for managing my television shows.

NZBDrone is a program designed to

manage a local collection of television shows. It also has the capability of working with Usenet programs to automate the possibly illegal downloading of episodes, but that's truly not all it's good for. NZBDrone will take your TV show files and organize them into folders, download metadata and let you know if you're missing files. It also will show you

when your favorite shows are going to be airing next.

Although it hasn't given me a problem, the fact that NZBDrone runs on Mono makes me nervous. The installation guide on the <http://www.nzbdrone.com> Web site makes setup simple enough, but there will be a boatload of dependencies that you might have to install due to its Mono infrastructure.

NZBDrone will work with your existing Plex media server, XBMC machines and SABNzb installs, and

it automatically performs most of its features if you allow it to do so. The interface is beautiful, and even with a large collection of television shows (I have more than 15TB of TV shows on my server), it's very responsive. Whether you record your TV episodes, rip your television season DVDs or find your episodes in other ways, NZBDrone is a perfect way to manage your collection. It's so intuitive and user friendly, that it gets this month's Editors' Choice award!

—SHAWN POWERS

LINUX JOURNAL

now available
for the **iPad** and
iPhone at the
App Store.



linuxjournal.com/ios



For more information about advertising opportunities within *Linux Journal* iPhone, iPad and Android apps, contact John Grogan at +1-713-344-1956 x2 or ads@linuxjournal.com.

URLs



REUVEN M.
LERNER

How URLs, a technology that we take for granted, are changing with the times.

The world Wide Web recently celebrated its 25th anniversary. As I have written in previous columns, the growth and ubiquity of the modern Web never cease to amaze me. I get my news, television and podcasts via the Web, not to mention my groceries and airline tickets, and it allows me to and communicate with my consulting clients around the world.

From my perspective, part of the genius of the Web, designed by Tim Berners-Lee, was its simplicity. Numerous researchers had been discussing hypertext for years before Berners-Lee appeared on the scene—and when he did, it was with a set of technologies that remain with us: HTTP, HTML and URLs.

I wouldn't claim that these technologies are unchanged after 25 years of usage, but it is pretty amazing to see how much they resemble their original versions. HTML has become, of course, far more sophisticated, thanks in no small part to the HTML5 set of standards, along with JavaScript and CSS. HTTP

has undergone a number of changes to improve its efficiency through the years, and it would seem that HTTP 2.0 eventually will be released, bringing with it considerably improved performance and security.

The lowly URL, however, has remained largely unchanged—at least, until now. In this article, I want to spend a bit of time talking about URLs (and their cousins, URIs and URNs) and the changes that are happening in the world of Web technologies. In particular, we're seeing changes in Web application technologies that have far-reaching implications for how we use the Web and for the ways in which our URLs function—especially as the Web becomes increasingly full of mobile and single-page apps.

Uniform Resource X

The idea behind URLs (Uniform Resource Locators) is a simple one: it identifies a document on the Internet. If you're reading this article, you presumably know that a URL can look like this: `http://example.com/foo`.

URLs are remarkably flexible, in that they can specify any protocol—and then for each protocol, a URL can specify a particular access method.

The first URLs were defined in fairly simple terms. There was a protocol, followed by a colon, and then (in the case of HTTP, at least) a server name, port number and pathname.

But soon after URLs first were unveiled, people started to consider that other things deserved unique identifiers. For example, let's say you want to refer to a book. Each book has a unique ISBN, so shouldn't it be possible to refer uniquely to a book via its ISBN? The IETF, which is in charge of many Internet standards, certainly thought so, and thus created the idea of a URN, or Uniform Resource Name. Whereas a URL points to a resource on the Internet, via a protocol, server name and pathname, URNs point to off-line resources via a unique code. Thus, you can point to a book with `urn:isbn:0451450523`.

Where does this book reside on the network? That's not the sort of question you're supposed to ask about a URN. URNs uniquely identify resources, but they don't tell you where to find them on-line. A URN always begins with "urn:", followed by the type of resource you're

describing. Following that, you'll then have a unique identifier for that resource. A URN should be unique; many books may share the same title, but each book has a unique ISBN.

With the creation of URNs, both URNs and URLs then became specific types of URIs, or Uniform Resource Identifiers. A URI can be a URL, identifying a particular on-line location. Or a URI can be a URN, pointing to a unique resource in the world.

Parts of a URL

Although URNs certainly are a great idea, I haven't ever used them in my work. But I have used URLs extensively, and I expect that all other Web developers have done so too.

URLs are remarkably flexible, in that they can specify any protocol—and then for each protocol, a URL can specify a particular access method. The following URL, `http://lerner.co.il/`, thus indicates that the resource is available via the HTTP protocol. HTTP URLs then have a hostname, followed by an optional port number that defaults to 80 for HTTP and 443 for HTTPS (that is, HTTP with SSL encryption).

To date, the fragment probably has been the smallest and most easily ignored part of a URL. But that is changing, and rapidly, thanks to the rise of single-page applications.

So the previous URL also could be written as `http://lerner.co.il:80/`, but there's generally no need to do so. Following the slash that comes after the hostname, there is a path. So, I can say: `http://lerner.co.il/team`.

This is where things start to get a bit interesting. The `/team` is passed to the Web server at `lerner.co.il` and describes...well, we don't know what it describes. To the outside world, the `/team` path seems to indicate part of a hierarchy, and probably even a document. Inside the Web application, it can be anything at all. In modern Web applications, the "router" looks at the URL and decides which object and/or method should be activated based on the path.

Now, in most cases, this is all you're going to need. But there are some additional, often ignored parts of URLs that are becoming increasingly important. For example, the hash character (`#`) can exist in the URL, and it separates the main URL from the "fragment". What is a fragment? Whatever you want it to be—that part

of an HTTP URL is handled internally by the application and/or the browser.

For years, the fragment was used to let you skip to a particular part of a page. So if you went to `http://example.com/foo.html#section2`, and if there was a "name" link inside the page with the value `section2`, the browser would move you there.

Another use for the fragment was to provide a URL for links that didn't exist for actual linking, but rather so that JavaScript could fire. That is, you could create a link like:

```
<a id="click-me" href="#">Click me</a>
```

If you were to click on such a link from a browser without JavaScript, nothing would happen. But in a browser with JavaScript, the page presumably would set a callback, such that clicking on the link would fire up some JavaScript code.

To date, the fragment probably has been the smallest and most easily ignored part of a URL. But that is changing, and rapidly, thanks to

the rise of single-page applications. However, before I discuss those, let me first talk about REST and what it means for URLs.

REST

“REST” has nothing to do with sleep; it is an acronym for Representational State Transfer and was coined by Apache cofounder Roy Fielding in his PhD dissertation. The idea behind REST is that you often see URLs as ways to access applications and documents on the Web, including the things you want to do with those applications and documents. So you might have a `/register` URL on your site, as well as a `/view_status` or `/see_book?id=100`.

The REST says that you should stop creating such URLs, and that you should instead see a URL as a unique way to describe an on-line resource. Thus, user 100 on your system becomes `/user/100`. Wait, you want to do something with user 100? That requires a verb, rather than a noun. Instead of using the URL, or part of it, for the verb, you instead should use the verb that already is being used with the URL—namely, one of the appropriate HTTP verbs. Most of us are only familiar with the HTTP methods GET and POST, but there are a bunch of others too. (Not that they’re really supported by most

browsers, of course.)

Now, I must admit that when REST became a mainstream, and even preferred, way to create URLs with Ruby on Rails, I tended to resist it. But over time, I have learned to appreciate the elegant simplicity of these URLs, particularly in an age when a growing number of HTTP verbs are supported by browsers, or (as in the case of Rails) you can automatically provide a parameter that indicates the request method, overriding the POST that you always send.

Rails has been particularly successful at pushing REST as a paradigm, in that controllers are assumed to provide seven different methods automatically, which are mapped in a standard way to combinations of HTTP request methods and URL patterns. Now, just because Rails does REST a certain way doesn’t mean that everyone needs to do it in precisely that way, using the specific URL style and meaning that Rails has defined. But that style, or something very close to it, has become quite popular, as you can see from such packages as Grape API for Ruby or Django REST Framework for Python.

One of the interesting aspects of using URLs in a REST framework is that the URL now describes an object, which often is mapped not only to a router and/or controller, but also to

More recently, Backbone and its ilk have given way to a new and more thoroughly designed type of framework, with the two leading contenders being Ember.js and Angular.js.

an object in a database. Thus, the URL `/users/1` effectively will allow me to retrieve, via the Web, information about user with ID 1.

Although such information used to be passed in XML or even in HTML, it's now fairly standard to transmit API data using JSON, which is standard, easy to work with and implemented in all modern languages. A RESTful API that uses JSON is increasingly common as the browser portion of applications becomes more important and needs to load and save data using these APIs.

Single-Page Apps

The most recent version of a Web application is the single-page application. From a user's perspective, you can call it a "single-page application", because it doesn't ever need to refresh the whole page, even when you click on a link or a button. Rather, JavaScript changes the page on the fly, modifying the DOM elements and reacting to events within the browser window.

It's possible to create single-page

applications using a library such as jQuery, but as things get complex, it becomes somewhat difficult and frustrating to do so. You end up spending time developing solutions that handle the infrastructure of such an application, rather than the application itself. If this sounds familiar, that's because the same thing happened about a decade ago. People were tired of writing the same code again and again for their Web applications. As a result, the notion of a "framework" was born, with Rails and Django being two of the most prominent players in that space.

Backbone.js was one of the first client-side frameworks, but it wasn't the only one. Indeed, there are dozens of frameworks, each claiming to be some degree of MVC (model-view-controller) that run in the browser and allow programmers to create rich, client-side applications in relatively short order. More recently, Backbone and its ilk have given way to a new and more thoroughly designed type of framework, with the two



DAVE TAYLOR

Considering Legacy UNIX/Linux Issues

Updating his shell script book, Dave bumps into a legacy Solaris problem, which leads to all sorts of interesting solution paths and discussion.

Gah, so frustrating! Ten years ago I wrote a rather popular book called *Wicked Cool Shell Scripts*, and I'm working on a new edition—a Tenth Anniversary release. There are lots of new scripts, entirely new chapters and updates to the older stuff. Fortunately, Bash hasn't evolved that much in the last decade, so just about everything still works fine (although there are some scripts I'm now realizing can't handle spaces in filenames—something I talked about years ago in this very column).

But, there were problems when I pushed out the following script to my Google Plus followers (find me on G+ at <http://profiles.google.com/d1taylor>) and asked those that had access to a Linux or UNIX system to give it a quick run:

```
#!/bin/sh

# how many commands: count how many executable commands
#   are in your current PATH.

myPATH="$(echo $PATH | sed -e 's/ ~/ /g' -e 's/:/ /g')"
```

```
count=0 ; nonex=0

for dirname in $myPATH ; do
    directory="$(echo $dirname | sed 's/~/ /g')"
```

```
    if [ -d "$directory" ] ; then
        for command in $(ls "$directory") ; do
            if [ -x "$directory/$command" ] ; then
                count=$(( $count + 1 ))
```

```
            else
                nonex=$(( $nonex + 1 ))
```

```
        fi
    done
fi

echo "$count commands, and $nonex entries that weren't \
    marked executable"
```

```
exit 0
```


It's simple enough really—using `sed` to split the `$PATH` value into space-separated values, then the `for` loop to step through them one by one, counting how many entries are marked as executable (the `-x` test).

Of course, you have to take into account that there might be spaces in directory names within the `PATH` (like `/User Applications/bin`), so I also convert spaces to `~~` and then later in the `for` loop convert them back at the last possible moment. But that's not rocket science, just basic scripting.

Why test to see if the directory in the `PATH` is an actual directory (the `-d` test), you may ask? Because when people can add their own directories to the system `PATH`, it can get messy, and it's entirely possible that there is an entry that's not a valid directory. So that's just error management really. Perhaps an `else echo "Error: Entry $directory isn't a directory?"` would be a good addition.

In any case, I posted this script, and people ran it on various systems, reporting answers ranging from 1,100 to more than 3,000 executable commands in their `PATH` (Ubuntu 13.10). More than 3,000 commands? Sheesh! Except then there was my friend Chris who said:

Sun OS 5.8 The line “`for command...`” gives me this error “syntax error at line 10: ‘`$`’ unexpected”.

That's this line in the script:

```
for dirname in $myPATH ; do
```

Well, that's puzzling, because there's nothing particularly complicated in that statement. Perhaps it's the `;` in the middle of the line? Still, a classic—useless—error message from the shell. A bit of digging, and it turns out that he had a different default login shell, and that `/bin/sh` in that version apparently wasn't linked to `/bin/bash`. Oops. We changed the first line to invoke the proper shell:

```
#!/bin/bash
```

And...it still didn't work:

The script ran but it came back with this: “First RE may not be null”. The second line read, “0 commands, and 0 entries that weren't marked executable”. There are a lot of executable files in my path.

Well heck. Now what?

Running Solaris to Test the Script

The logical solution was to gain access to a system running Solaris (ideally SunOS 5.8, aka Solaris 8), but who the heck is running a Solaris system and can grant me external SSH access?

The answer: no one I could find, which is why it's fortunate that I found a far better path: VirtualBox.

Free to download from Oracle (<https://www.virtualbox.org/wiki/Downloads>), VirtualBox is a virtualization system, creating a system within a system. Even better, it runs on Mac or Windows systems

along with various flavors of Linux, offering the ability to install and run a full Solaris installation (or just about any other OS you're interested in testing) as an app.

If you've experimented with VMware or Parallels, you've already bumped into this technology, and it's very slick. In fact, I run Windows 8 Pro on my MacBook Pro using VMware Fusion, and it works astonishingly well in its own full-screen window. The down side is that VMware Fusion isn't free. But, VirtualBox is—nice.

Download and install it, then

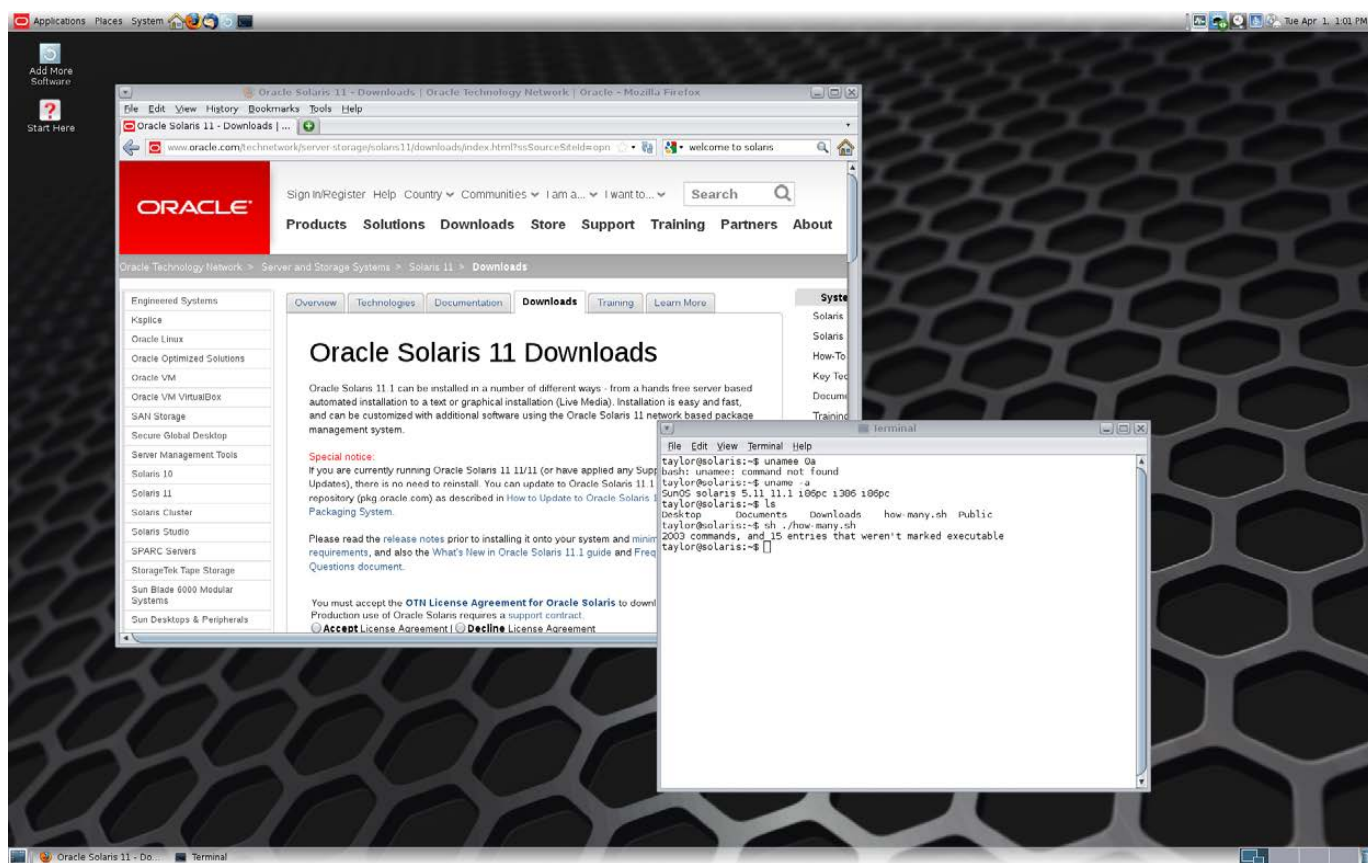


Figure 1. Solaris 11.1 Running within VirtualBox, within Mac OS X

you can grab a free copy of SunOS 5.11 (aka Solaris 11.1) at <http://www.oracle.com/technetwork/server-storage/solaris11/vmtemplates-vmvirtualbox-1949721.html>.

Unpack the OS and double-click. It's automatically opened by VirtualBox, and with another click or two, you're running Solaris 11.1 and have the default window manager, GNOME, front and center, as shown in Figure 1.

Now finally, I can open up an xterm and test the script within a Solaris environment. The easiest path? Tweak

the VirtualBox config to share the clipboard with the parent operating system, and you simply can copy and paste it into a vi edit buffer and save it.

An invocation:

```
$ sh ./count-cmds.sh
```

2003 commands, and 15 entries that weren't marked executable

Ah great. So in fact, the script works fine in the latest version of SunOS/Solaris but fails in the older version that Chris is running. How old is it? It turns out that Solaris 8 came out a while back, in February 2004.

LINUX JOURNAL

on your
Android device

Download the app now
in the **Android Marketplace**

www.linuxjournal.com/android



For more information about advertising opportunities within *Linux Journal* iPhone, iPad and Android apps, contact John Grogan at +1-713-344-1956 x2 or ads@linuxjournal.com.

This leads to the dilemma: the script apparently doesn't work on a ten-year-old version of Solaris UNIX but works just fine on the latest release, Solaris 11. Should I care?

This is all tied to the legacy problem: how far back do you need to go to ensure that your software works? The previous OS release? Five years back? Ten? Longer? Legacy support has been in the news for Windows users, that's for sure, as Microsoft just axed support for the ancient Windows XP version of the flagship operating system. For the record, WinXP was released in October *2001*. Fourteen years later, Microsoft is saying "guys, we've had a lot of major releases since then and can't support it forever", and people are howling.

Apple seems to weather this sort of thing more gracefully. When the company moved from MacOS to Mac OS X, it included “Classic Mode” where old apps would mostly run, but the writing was on the wall from the beginning of the OS X era that Apple wasn’t going to “do a Microsoft” and support the old OS for years and years.

And, this brings me back to

Solaris 8 and *Wicked Cool Shell Scripts*. The long and short of it: if the script didn't work properly in Solaris 11, I'd be concerned and debug the problem, but because it fails in a ten-year-old version of the OS, I'm going to ignore the problem. If I could log in to a Solaris 8 system, I might debug it anyway just to understand what's going on, but is that a reason to slow down the revision of the book? I don't think so.

Legacy support—it's a big challenge for every software developer, and although Bash and the Linux command-line world hasn't changed that much in the past few years, it's still something to consider before you ship your own software (even if it's free software).

So what's your solution? Write to us, and let us know how your company deals with legacy Linux/UNIX issues! ■

Dave Taylor has been hacking shell scripts for more than 30 years. Really. He's the author of the popular *Wicked Cool Shell Scripts* and can be found on Twitter as @DaveTaylor and more generally at his tech site <http://www.AskDaveTaylor.com>.

|||||

Send comments or feedback via
<http://www.linuxjournal.com/contact>
or to ljeditor@linuxjournal.com.

REGISTER TODAY!

2014 USENIX Federated Conferences Week

Cloud, Storage, Sysadmin, and More

June 17–20, 2014 Philadelphia, PA

www.usenix.org/fcw14

Back for 2014, USENIX is combining established conferences and workshops into a week of research, trends, and community interaction. Events include:

USENIX ATC '14

2014 USENIX Annual Technical Conference

Thursday–Friday, June 19–20 www.usenix.org/atc14

ICAC '14

11th International Conference on Autonomic Computing

Wednesday–Friday, June 18–20 www.usenix.org/icac14

Feedback Computing '14

9th International Workshop on Feedback Computing

Tuesday, June 17 www.usenix.org/feedback14

HotCloud '14

6th USENIX Workshop on Hot Topics in Cloud Computing

Tuesday–Wednesday, June 17–18 www.usenix.org/hotcloud14

HotStorage '14

6th USENIX Workshop on Hot Topics in Storage and File Systems

Tuesday–Wednesday, June 17–18 www.usenix.org/hotstorage14

UCMS '14

2014 USENIX Configuration Management Summit

Thursday, June 19 www.usenix.org/ucms14

URES '14

2014 USENIX Release Engineering Summit

Friday, June 20 www.usenix.org/ures14

WiAC '14

2014 USENIX Women in Advanced Computing Summit

Wednesday, June 18 www.usenix.org/wiac14

Training Sessions are back!

Topics include:

- Hadoop
- Apache Cloudstack
- Jenkins
- SRE
- Security
- Autonomic computing



www.twitter.com/usenix



www.usenix.org/facebook



www.usenix.org/youtube



www.usenix.org/linkedin



www.usenix.org/gplus



www.usenix.org/blog





KYLE RANKIN

Encrypt Your Dog (Mutt and GPG)

Like most common things with Mutt, encryption and signing of e-mail is fairly straightforward and customizable.

I have been focusing a lot on security and privacy issues in this year's columns so far, but I realize some of you may expect a different kind of topic from me (or maybe are just tired of all this security talk). Well, you are in luck. I'm going to kill two birds with one stone and describe security as applied to a piece of software that has gotten a lot of play in my column through the years: Mutt. Those of you who are familiar with my column know about my long history as a Mutt user. For those of you who aren't acquainted with it, Mutt is a command-line mail client (some would say Mail User Agent or MUA) that is highly configurable and uses vi key bindings that so many of us know and love. If you want an initial primer on Mutt, check

out my article "Take Mutt for a Walk" from the October 2010 issue (<http://www.linuxjournal.com/article/10858>). I've written a lot about Mutt in the past, but I realized recently that I never really discussed Mutt's PGP/GPG integration before now.

Mutt PGP/GPG Settings

The first step in the process is to configure Mutt's PGP/GPG settings. Actually, the first step probably should be for you to create a GPG keypair if you haven't already, but that is a topic for another article and one that's already been covered in *Linux Journal*. Mutt has quite a few settings for PGP, but in my experience, you need to be concerned about only a few. So, add the following lines to your `~/.muttrc` file, and I will discuss each of the

Once your PGP settings are in place, Mutt automatically should sign or encrypt replies to encrypted or signed messages in a common-sense way.

options in detail:

```
set pgp_replyencrypt      # now crypt_replyencrypt
set pgp_replysign         # now crypt_replysign
set pgp_replysignencrypted # now crypt_replysignencrypted
set pgp_show_unusable=no
```

The first thing to notice (and something I didn't realize until I was writing this article) is that Mutt's development release has changed the name of some of these settings. All of the encryption settings used to be prefaced by `pgp_`, but now some of the settings have been abstracted out presumably to work with things other than PGP and begin with `crypt_` instead. In my experience, the old setting names still work, and as that has the widest compatibility, I refer to the settings by those names.

The first three settings enable what I consider to be sane defaults for encrypted e-mail. Although Mutt has a series of settings that let you automatically sign and encrypt every message you send (they all start with `pgp_auto` or `crypt_auto` if you are

curious), that probably isn't practical for most people. Instead, these `pgp_reply` settings configure how to behave when you reply to a message that has been signed or encrypted.

The `pgp_replyencrypt` setting automatically will encrypt replies to encrypted messages, and `pgp_replysign` automatically will sign messages that have been signed. If a message has been signed and encrypted, the setting `pgp_replysignencrypted` takes care of automatically encrypting and signing replies. The final setting, `pgp_show_unusable=no`, will hide any PGP keys in your keychain that have expired, have been revoked or are otherwise unusable.

Use PGP/GPG inside Mutt

Once your PGP settings are in place, Mutt automatically should sign or encrypt replies to encrypted or signed messages in a common-sense way. Of course, that doesn't help with conversations you want to start, or if you want to encrypt or sign a reply to a message that isn't encrypted.

Mutt makes it easy to change the security status of any message before you send it. After you compose and save a message, you will be on a screen that shows you the To, CC, From and Subject for the message. This is the same screen where you would add any attachments and where you press the y key to send the message. The Security field on this screen shows your current PGP settings for the message. If you haven't enabled signing or encryption for the message, this field will be set to None. Otherwise, it might be set to Sign or Encrypt or Sign, Encrypt. To change your security settings, press p and then select from encrypt (e key), sign (s key), sign as (a key), both sign and encrypt (b key) or clear (c key), which disables any security settings. If you choose to encrypt the message, when you send it, Mutt will present you with recipient encryption keys from which to choose.

Mutt PGP/GPG Hooks

Of course, you could enable PGP signing or encryption manually on a per-message basis, but you might have a friend or colleague that you know uses e-mail encryption and to whom you always want to sign or encrypt your messages. In that case, Mutt provides hooks to allow you to configure when to enable security settings automatically.

Let's assume I wanted to sign all

messages I send to linuxjournal.com, but I specifically wanted to sign and encrypt messages sent to editor@linuxjournal.com. I would add the following settings to my ~/.muttrc:

```
send-hook . 'unset pgp_autosign; unset pgp_autoencrypt'
send-hook '~t @linuxjournal.com' 'set pgp_autosign'
send-hook '~t editor@linuxjournal.com' 'set pgp_autosign;
set pgp_autoencrypt'
```

The send-hook setting allows you to configure Mutt settings that apply right before you send a message. The syntax with Mutt hooks is send-hook followed by a pattern, then followed by one or more settings. The initial line:

```
send-hook . 'unset pgp_autosign; unset pgp_autoencrypt'
```

is set to match all messages (the . matches anything). It then unsets any automatic signing or encryption. This acts as your default setting, and it's important that it appears before any other PGP-related send-hook lines. This default exists so that if you trigger any other send-hooks and enable automatic signing or encryption when sending to a specific address, this hook will unset it before you send a message to someone else.

The next line will sign any messages sent to linuxjournal.com automatically:

```
send-hook '~t @linuxjournal.com' 'set pgp_autosign'
```

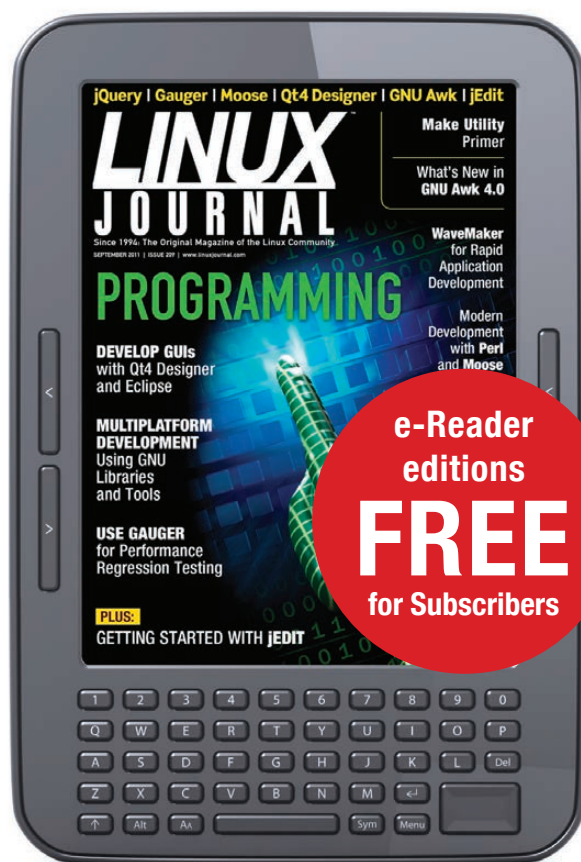
With these settings in place, you should be able to feel safe knowing

Kyle Rankin is a Sr. Systems Administrator in the San Francisco Bay Area and the author of a number of books, including *The Official Ubuntu Server Book*, *Knoppix Hacks* and *Ubuntu Hacks*. He is currently the president of the North Bay Linux Users' Group.

|||||

Send comments or feedback via
<http://www.linuxjournal.com/contact>
or to ljeditor@linuxjournal.com.

LEARN MORE





SHAWN POWERS

Being a Hack

What's better than chocolate and peanut butter? Bash scripts and FOR loops!

If you think hacking is breaking into Pentagon computers to play “Global Thermonuclear War” with Joshua, you have good taste in movies, but unfortunately, not a clear picture of what hackers do. Yes, there is a subset of folks who take advantage of system vulnerabilities to compromise computer systems. There’s a much larger group of people, however, who just use quick bits of code to get their jobs done. These “hacks” aren’t nefarious, but are generally not well planned and executed code. Hacks are like digital duct tape, and although you probably can hold an airplane’s wing on with duct tape, you wouldn’t want to fly it very far. The same is true with the hacks I talk about here. They’re generally good for a quick fix, but not something you want to build your infrastructure on. (Unfortunately, simple hacks often get hacked on more and more, and become production systems, which is not ideal, but nonetheless can happen. Use your hacking powers wisely and know when your digital duct tape isn’t appropriate.)

In my last article, I described a bunch of simple skills that I now want to demonstrate in action. Basically, I’m just going to think up a handful of things I’ve done through the years and show you an example script. Then I’ll go through them. You probably won’t have the same needs I do, but hopefully the concepts will get you thinking. For example, let’s start with a script I used to use on my file server to create home directories for newly added users. On a standalone system, the home directories are created when you add a user, but on a large network, the processes are often separate. My users would get added to an LDAP database, and then I’d run the following to create their home directories:

```
#!/bin/bash
#
# Create home directories on file server

cp -R /etc/skel /home/$1
chown -R $1.$1 /home/$1
chmod 751 /home/$1

echo "Unless you saw an error, everything is good."
```


If you remember from my last article, the `$1` variable is filled with the first argument given to the script. In this case, it's a user name (like "spowers"). The script then copies the `/etc/skel` folder and all of its contents to `/home` with the name of the new user. Then ownership is changed to the user's user name and group, and finally the permissions are set on the user's folder. In my case, it allows non-owners to enter the directory so Apache can read the user's public Web folder. This is a real-world example of how to use the `$1` variable. If you had a separate group, you could use `$2` to specify. Building on this example, you can come up with elaborate variations to suit your needs.

Programs Never Die

The next script is far smaller, but it serves an interesting purpose. If you're running a program that is known to crash occasionally (Mono programs are notorious for this, at least in my experience), it's helpful to have them automatically restart. It's possible to create an init script or upstart configuration that will respawn dead processes, but it's often challenging to get the configuration just right. A quick hack is to put the program in an

endless loop. Here's an example:

```
#!/bin/bash
#
# Restart program when it dies
while true
do
    /usr/bin/crashy_program
    sleep 10
done
```

If you start this script from `rc.local` or from `crontab` on boot (see my article in the UpFront section of this issue on using cron to start programs at boot), it will run `crashy_program` until it crashes, wait 10 seconds, and then the loop starts over, which launches the program again. You do have to be careful, because if "crashy_program" is something that launches itself into the background, in `dæmon` mode, the script will just keep starting new instances of the program until your RAM fills up. You could add a `kill crashy_program` line inside the loop if you want to clean up any remaining processes before starting the loop over, but this method of keeping a script running will work well only if the program doesn't release control of the shell while it's running. Hopefully that makes sense.

Can You Hear Me Now?

From here out, the scripts I show you will be more and more complex. There might even be some stuff thrown in that I didn't cover in my last article, but that's okay, it should be pretty easy to figure out what's going on. Take this script for example, which I use to check my Internet connection at home:

```
#!/bin/bash

#

# Test Google by IP

wget -q --tries=3 --timeout=5 \

    http://173.194.46.49 -O /tmp/google.idx &> /dev/null

if [ ! -s /tmp/google.idx ]

then

    /usr/local/bin/powercontrol reboot

    sleep 180

    echo "Charter sucks." | mail -s \

        "DANGER WILL ROBINSON: Rebooted Home Router" me@example.com

fi

rm -rf /tmp/google.idx
```

This is literally the code I use to check my Internet connection and power cycle my modem if need be. First things first, the backslash in a script is just a way of making the commands more readable. All the `\` character does is break a single command into multiple lines. The

system doesn't actually see the `\` character, it sees the entire line. So above, the `wget` command is a one-liner that ends with `/dev/null`.

The script itself uses `wget` to download the Google search page to `/tmp/google.idx`. I use an IP address because often when my modem is off-line, DNS lookups fail, so that IP address is one of Google's. Anyway, `wget` tries to download the Google page, allowing for three failed attempts with a five-second timeout. Then the "if" statement checks to see if it failed at downloading the file. (That's what the `!` does, it negates the test command.) If it failed, it issues a reboot command to my serial-port-connected power-cycling machine, waits three minutes for the connection to come back up, and then e-mails me a notification of the failure. If `wget` successfully downloads the file, which it usually does, the if statement is skipped, the downloaded file is erased and the script ends. I run this via cron every 15 minutes or so, and it works well to keep my flaky connection stable.

Everyone Gets a Web

The next script goes back to the home directory situation. This time, however, I use a "for" loop to affect change to all the folders in the `/home`

directory. See if you can figure out what this does:

```
#!/bin/bash
#
for x in `ls /home`
do
mkdir /home/$x/public_html
chown $x.nobody /home/$x/public_html
chmod 755 /home/$x/public_html
done
```

This script basically creates a set of objects from the `ls /home` command (because it's in backticks), and then executes one loop iteration for each object in the set. The beauty of this is that it will work whether you have three users or 3,000 users. Each iteration of the loop (the part between `do` and `done`) creates a `public_html` folder inside the user's folder and gives it the correct ownership and permissions. You can imagine how much typing this saves for large numbers of users! I use a variation on this type of loop for lots of maintenance issues on user files. If I need to copy a single file to everyone's desktop, a `for` loop saves the day.

This is probably a good time to remind everyone that quick Bash hacks like these aren't foolproof. It's best if you first have your script do something innocent like `echo` instead of `mkdir`,

so that it prints on the screen what it is doing. A simple typo could cause you to wipe out millions of user files, so it's best to test your script before using it on your live servers or personal system. This is especially true if you start running `rm` commands in a loop—that's some powerful mojo, which you don't want to use incorrectly.

I Hate Typing Things More Than Once

Finally, I'm going to demonstrate another way I use quick Bash scripts on a regular basis, and that is to create configuration files. Basically, any time you see repetitious data in a configuration file, chances are you can write a script that will save you lots of time. This script is fairly complex, but it uses lots of the tools I've been talking about. This configuration file is actually part of a script I use to monitor Bitcoin miners, for those who are curious:

```
#!/bin/bash
#
BASE_ADDRESS="172.20.1."
LOOP_NUMBER=$(( $2 - $1 ))
# First part of config file
echo "<?php"
# This loop should run for all miners
for MINERLOOP in $(seq 0 $LOOP_NUMBER);
do
echo "\$r[$MINERLOOP]['name'] = 'MINER$(( $1 + $MINERLOOP ))';"
```

```
echo "\$r[$$MINERLOOP]['ip'] = '$BASE_ADDRESS$((($1 + $MINERLOOP))';"
echo "\$r[$$MINERLOOP]['port'] = '4028';"
echo "\$r[$$MINERLOOP]['sick'] = 'FALSE';"
echo " "
done

# And finish off the file
echo ">"
```

To make this program run, it needs two arguments. The last octet of the IP addresses of the miners I'm configuring must be entered, so I'd type something like:

```
./myscript 100 102
```

And the output is:

```
<?php
$r[0]['name'] = 'MINER100';
$r[0]['ip'] = '172.20.1.100';
$r[0]['port'] = '4028';
$r[0]['sick'] = 'FALSE';

$r[1]['name'] = 'MINER101';
$r[1]['ip'] = '172.20.1.101';
$r[1]['port'] = '4028';
$r[1]['sick'] = 'FALSE';

$r[2]['name'] = 'MINER102';
$r[2]['ip'] = '172.20.1.102';
$r[2]['port'] = '4028';
$r[2]['sick'] = 'FALSE';

?>
```

If you follow the logic of the script, you'll see it starts by figuring out the number of loops needed by subtracting the beginning IP octet from the ending one—in this case, $102-100=2$. You'll notice there are actually three iterations of the loop, and that's because I'm a little sneaky. I start the loop iterations at zero, so there are three total loops done. Little quirks like this are figured out as you test your scripts and are the reason you *must* test your scripts before depending on them, even if they're for something simple like this.

Anyway, there are some confusing things in this script that I had to learn how to do while I was debugging it originally. The `$(seq 0 $LOOP_NUMBER)` statement, for example, is really confusing looking. The reason it's required, however, is because it's not possible to put a variable in a standard range statement for creating a for loop. My first instinct was to say `for MINERLOOP in {0..$LOOP_NUMBER}`, but that just doesn't work. My brain thinks it should work, but alas, it doesn't. So, using the `seq` command along with the `$()` structure provides the same effect, only with `seq`, it works.

There might be some confusion with the `echo` statements too,

```
./myscript 100 102 > config.php
```

You've Ruined My Mental Image of Sysadmins

I know some of these scripting examples seem absurdly simple. Some of the most useful scripts are! The idea with this article was to get you thinking about how to combine the various scripting basics into something powerful, something useful and ultimately something that saves you time. ■

He's also the Gadget Guy for LinuxJournal.com, and he has an interesting collection of vintage Garfield coffee mugs. Don't let his silly hairdo fool you, he's a pretty ordinary guy and can be reached via e-mail at shawn@linuxjournal.com. Or, swing by the [#linuxjournal](https://www.freenode.net) IRC channel on [Freenode.net](https://www.freenode.net).

WWW.LINUXJOURNAL.COM / JUNE 2014 / 51

IGEL Thin and Zero Clients

Those working in the clinical field will want to take note of innovations to the IGEL Thin and Zero Client solutions. IGEL Technology recently announced that all Linux x86-based hardware and software thin clients now integrate Imprivata OneSign ProvelD Embedded, providing the health-care industry with secure clinical authentication.



The addition of OneSign ProvelD Embedded allows high-performance, user authentication for rapid, secure access to clinical applications and critical patient information. Instead of having to enter passwords manually, users log on to virtual desktops and/or communicate through virtual channels to core medical applications using a contactless RF-enabled smart card. The resulting improvement of work processes ensures that staff members have more time to spend with patients.

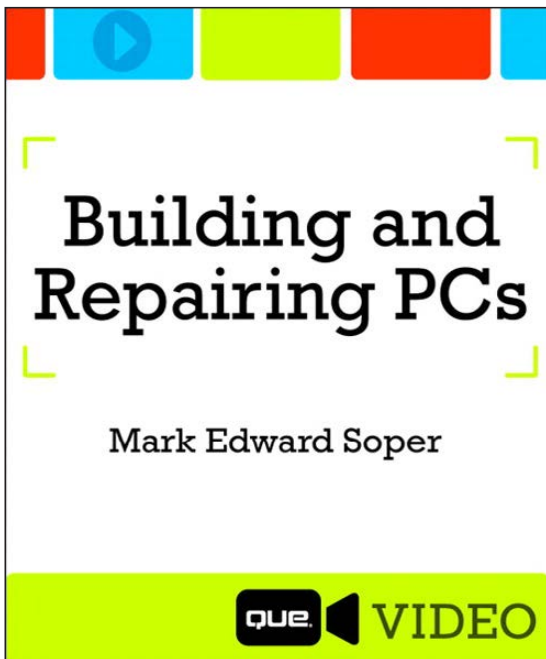
<http://www.igel.com>

elastichosts

ElasticHosts' Elastic Containers

In a test analysis with its largest customers, ElasticHosts found that switching to the company's new Elastic Containers—cloud servers that are based purely on consumption, rather than capacity—would result in a 50% or more cost savings. The secret sauce is a breakthrough auto-scaling container technology that elastically expands and contracts to meet customer demands, entirely eliminating the need for manual provisioning. Because each container automatically can scale up to 64GB RAM, companies can handle usage peaks and valleys more effectively. Aside from disrupting the cloud market, ElasticHosts adds that Elastic Containers also will eliminate load balancers and reduce disaster recovery costs.

<http://www.elastichosts.com>



Mark Edward Soper's *Building and Repairing PCs* (Que Video)

We Linuxers are a roll-your-own kind of crew. Even for us though, keeping track of all the latest technologies can be overwhelming. A solution with our tribe in mind is Mark Edward Soper's *Building and Repairing PCs*, a hands-on tutorial dedicated to building and repairing desktop and laptop PCs. The product consists of three hours of professionally edited, downloadable videos that cover topics like

selection of parts, building a system, tweaking the BIOS, overclocking, installing the latest components (such as hard disks), configuring a PC for game performance and more.

<http://www.informit.com>

Oswald Campesato's *Google Glass Development* (Mercury Learning)

Oswald Campesato's new book *Google Glass Development* adds to Mercury Learning & Information's newly developed Pocket Primer series. Campesato's book provides an overview of the major aspects, the source code and tutorial videos to develop applications for Google Glass. It also contains information for developing Glassware using Android and HTML5 technologies, primarily for self-directed learners who have some knowledge of Android and HTML5 graphics-related technologies. Other topics include CSS3, HTML5 Canvas, D3 and SVG, as well as the Glass GDK and working with sensors.

<http://www.merclearning.com>



Envivio's G5 Family of Appliances

envivio®

Envivio's specialty is software-based video processing and delivery solutions. The company's latest innovation is the Envivio G5 family of Intel-based server appliances, featuring increased compression density, support for the latest Ultra HD 4K resolution and HEVC (H.265) encoding, and a lowering of operating expenses for service providers. Operators deploying Envivio Muse Live encoders on the new G5 platform can deliver up to 100 high-quality SD or 20 HD channels in a 2RU configuration. In a typical IPTV or cable scenario, this represents a significant cost savings. Rack space requirements can be reduced by nearly 40% and power consumption by more than 30% compared to the previous generation of Envivio appliances. Both 1RU and 2RU versions are available, with the latter featuring a modular hot-swappable, multi-node architecture.

<http://www.envivio.com>



Elecsys e-Modem

Industrial equipment manufacturers that need a practical and reliable solution for linking their products with data networks will be pleased to learn about the novel Elecsys e-Modem series of embedded cellular data modems. These are wireless communication devices that are pre-certified to operate on the Verizon Wireless network and are ready to integrate into industrial products to add M2M connectivity into remote field applications and rugged equipment. Key product features include multiple cellular technologies (CDMA 1xRTT and EV-DO), Verizon Wireless Open Development certification and a design for industrial applications. Target applications include oil and gas wells, energy distribution systems, agricultural facilities, transportation infrastructure and many other industrial applications.

<http://www.elecsyscorp.com>



Red Hat Enterprise Linux

Thanks to collaboration with Google, users of Red Hat Enterprise Linux now have a new “bring-your-own-subscription” benefit that enables them to move their on-site subscription to the Google Compute Engine. The benefit is part of the Red Hat Cloud Access program, which enables Red Hat customers to take advantage of the benefits of the Google Cloud Platform with the confidence that the consistency and quality on a public cloud matches the on-site version. Red Hat states that Red Hat Cloud Access also enables customers to maintain a direct relationship with the company—including the ability to receive full support from Red Hat’s Global Support Services organization—on Google Compute Engine. This enables its customers to maintain a consistent level of service and support across all certified deployment infrastructures with consistent and predictable pricing.

<http://www.redhat.com>

Verocel’s VeroTrace



The new commercial version of Verocel’s VeroTrace, an advanced life-cycle management environment, features an enhanced architecture, an Eclipse IDE and a wealth of other capabilities. The upshot is that software developers now can automate the many tasks and processes required for their own large-scale, advanced software development and verification efforts. VeroTrace provides not only full traceability, review and workflow tracking but also monitors development and certification life-cycle artifacts as well as their relationships and authorization statuses. It is the ideal environment to facilitate the development, review, authorization and sign-off status of complex software systems, and aid in their delivery. VeroTrace already has successfully delivered safety and security projects and certification evidence to meet the DO-178B/C Avionic Software standards, the EN 50128 rail certification and the IEC 61508 for the industrial sector.

<http://www.verocel.com>

Please send information about releases of Linux-related products to newproducts@linuxjournal.com or New Products c/o Linux Journal, PO Box 980985, Houston, TX 77098. Submissions are edited for length and content.

Monitoring Android Traffic with Wireshark

Use some simple Linux tools and a laptop to get access to the Internet traffic sent and received by your smartphone.

BRIAN TRAPP

The ubiquity and convenience of smartphones has been a real boon for getting information on the go. I love being able to jump on a Wi-Fi hotspot, catch up on my mail, check my banking balance or read the latest tech news—all without having to bring along or boot up a laptop. Now that mobile development is mainstream, most of this access is done via specialized apps, instead of via a Web browser.

This migration away from direct Web access in favor of dedicated smartphone apps has made for a richer user experience, but it also has made knowing exactly what is going on “under the hood” a lot harder. On our Linux boxes, there are many tools to help user peer into the internals of what’s going to and from the machine. Our browsers have simple HTTP versus HTTPS checks to see if there’s encryption, and there are simple but easy-to-use browser plugins like Firebug that let us view exactly what’s being sent and retrieved over the Web. At the operating system level, powerful tools like Wireshark let us drill down even further, capturing *all* traffic flowing through a network interface. Smartphones usually are locked up to a point where it’s almost impossible for a regular user to run

any network monitoring or tracing software directly on the phone—so how can a curious user get access to that phone traffic?

Fortunately, with just a little bit of work, you can use Linux to transform almost any laptop into a secret-sharing wireless access point (WAP), connect your phone and view the data flowing to and from the phone with relative ease. All you really need is a laptop running Linux with one wireless and one Ethernet connection.

Intercepting Traffic

The first step is to set up your own “naughty” WAP where you can capture and log all the Internet traffic passing through it—simulating the kind of information that a rogue employee could be obtaining from a coffee-shop Wi-Fi hotspot. Let’s do this in a distribution-independent way that doesn’t mess around with your existing router (no need to change security settings) and doesn’t require rooting or installing anything unseemly on your phone.

To turn a laptop into a WAP, you’ll first use `hostapd` to use the wireless card as an access point mode (broadcasting an SSID, authenticating with security and so on). Next, you’ll

FALSE STARTS

It may be tempting to try a shortcut for capturing this traffic. Here are a few techniques I tried and discarded before sticking with a hostapd/dnsmasq/iptables solution.

UBUNTU'S BUILT-IN HOTSPOTS:

Ubuntu has a handy “Use as Hotspot” feature tucked away in its networking settings. Unfortunately, it creates hotspots in ad hoc mode, which isn’t compatible with most versions of Android. I didn’t try Fedora’s implementation, but the method I recommend instead will work on any distribution.

MONITOR MODE:

It’s tempting just to put the wireless card in monitor mode and capture *all* wireless traffic, independent of SSID. This is pretty cool, but there are quite a few “gotchas”:

- The drivers for your wireless card must support monitor mode. Many, but not all cards support this mode.
- Your capture needs to include the four WPA “handshake” packets.
- You’ll probably have to compile and use airmon-ng to start monitor mode and then capture on the mon0 pseudo-device airmon creates.
- If the WAP is using encryption, the packets you capture also will be encrypted. Wireshark does have a facility to help decode the packets, but you’ll need to enter information about the security scheme used by the WAP and toggle a few sets of options until the decoded packets look right. For a first-time user, it’s hard enough making sense out of Wireshark dumps without having to worry about toggling security options on and off.

CAPTURING WITH THE ANDROID EMULATOR:

Another approach would be to use an Android emulator on your capture device, install and then run the target application, and capture the traffic from the emulator. It’s much harder than it sounds actually to get a banking app on the emulator though:

- Due to recent Android licensing changes, the major Android VMs no longer include the Google Play store. (I tried both the Android SDK and the free product from Genymotion.)
- If your phone isn’t rooted, it’s not easy to get the application’s .apk off your phone and onto the VM.

use dnsmasq to provide DNS and DHCP services for clients connecting on the wireless connection. Finally, iptables' masquerading features will be used to direct IP traffic from clients on the wireless connection to the Internet (via your Ethernet connection), and then rout responses back to the correct client on the wireless side.

hostapd

hostapd is a small utility that lets you create your own wireless access point. Installation is straightforward, and configuration is just as easy. Most wireless cards and modern kernels will be using the mac80211 driver. Check yours via `lsmod | grep mac80211`. If that's your driver, find your wireless device via `ifconfig`, and set up the SSID of your choice as shown below for an unsecured, totally open access point:

```
===[/etc/hostapd/hostapd.conf]=====  
interface=wlan0  
driver=nl80211  
ssid=WatchingU  
channel=1  
===[/etc/hostapd/hostapd.conf]=====
```

I recommend not using Wi-Fi security for this test; it would be overkill, as your access point will

only be temporary. Should you desire a more permanent solution, hostapd supports many different authentication options.

dnsmasq

Now that hostapd is ready to start letting clients connect to your wireless connection, you need dnsmasq to serve DHCP and provide DNS for your access point. Fortunately, dnsmasq is also very easy to install and configure. The example below is the minimum required. Make sure the `dhcp-range` you specify will not conflict with anything already on your network. By default, dnsmasq will read your existing `/etc/resolv.conf` and propagate the DNS settings listed there to its clients. That's a pretty sane default configuration, but if you need something else, use the `no-resolv` option and specify the DNS servers manually:

```
=====[/etc/dnsmasq.conf]=====  
interface=wlan0  
dhcp-range=10.0.0.3,10.0.0.20,12h  
=====[/etc/dnsmasq.conf]=====
```

iptables

The final piece of your wireless access point is iptables, which will use IP Masquerading to get the traffic from the wireless connection, send it over

the wired connection and route any responses back to the correct source on the wireless side. There are many distribution-specific ways to save and script iptables rules, but it's simpler to create a distribution-independent shell script to enable iptables and network address translation (NAT). A script for iptables that ties in hostapd and dnsmasq would look like the following (modify the wlan0 and eth0 entries to match your system):

```
=====[makeWAP.sh]=====

#!/bin/bash

export DEV_IN=wlan0;
export DEV_OUT=eth0;

echo "Bringing up $DEV_IN"
#This address/mask should match how you configured dnsmasq
ifconfig $DEV_IN up 10.0.0.1 netmask 255.255.255.0

echo "Starting dnsmasq"
dnsmasq

echo "Configuring iptables"
#Clear everything in iptables
iptables -Z;
iptables -F;
iptables -X;

#Turn on iptables NAT, forwarding, and enable
#forwarding in the kernel
iptables --table nat --append POSTROUTING --out-interface
```

```
➡$DEV_OUT -j MASQUERADE

iptables --append FORWARD --in-interface $DEV_IN -j ACCEPT

➡sysctl -w net.ipv4.ip_forward=1

echo "Starting hostapd"

hostapd /etc/hostapd/hostapd.conf 1> /dev/null

=====[makeWAP.sh]=====
```

To test everything, connect your capture laptop to a wired connection with Internet access and disconnect any existing wireless connections. Run the makeWAP.sh script (sudo ./makeWAP.sh) to start up the WAP.

On the phone, turn off mobile data (for Android 4.3, this is done via Settings→Data Usage→Mobile data→Off), turn on Wi-Fi, and connect to the new WAP (in the example above the SSID would be "WatchingU"). Once connected, test a few sites to make sure you can access data from the Internet.

If everything works, congratulations, you have transformed your laptop into the world's most ridiculously overqualified wireless router!

Wireshark

Wireshark is a network packet analyzer that you'll use to capture and make sense of the data flowing on your newly created access point. You'll be merely scratching the surface of its capabilities, as it is an

extremely powerful tool with abilities stretching well beyond “poke at a few packets” as used in this project.

Install Wireshark for your version of Linux. If at all possible, get version 1.10 or higher, as 1.10 adds support for decoding gzip’ed HTTP data on the fly (and there’s a *lot* of that). Prior to 1.10, you’d have to save the TCP stream to a file, edit out the header and then gunzip it to view the raw data. This becomes tedious quickly, so having Wireshark do all that for you behind the scenes is awesome.

When running Wireshark for the first time, if it complains that there are no devices available for capture, you have to give your ID permissions for the various devices and applications used by Wireshark. For Ubuntu, run `sudo dpkg-reconfigure wireshark-common`, and select the option to let nonroot users capture packets, and make sure your ID is in the “Wireshark” group. For other distributions, search for which devices and scripts need to be owned by which groups.

Before moving on to capturing traffic, shut down every non-essential app and service on the phone to make it easier to find the traffic of interest. The fewer packets you have to sort through, the better.

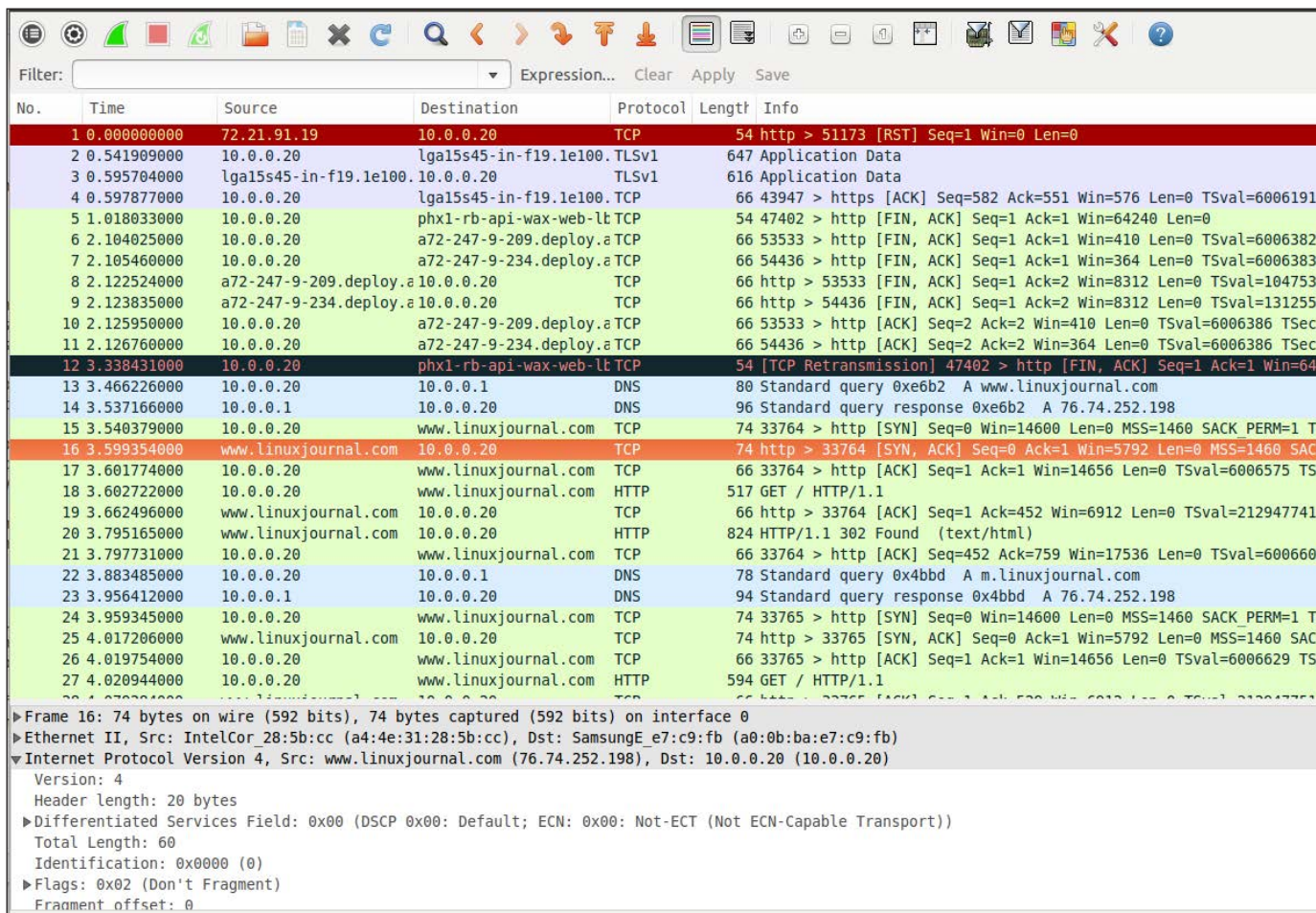
Capturing Unencrypted Web Traffic

Before you start looking for sensitive data, let’s first get familiar with what unencrypted traffic looks like in Wireshark.

- From the Wireshark starting screen, select the wireless device (wlan0) and then the “Start” icon to start a new capture.
- On the phone, use a browser to go to <http://www.linuxjournal.com>.
- Once the page finished loading on the phone, press the “Stop” icon in Wireshark, and save the capture file somewhere safe, called something like “Capture_LJ.pcapnp”.

Now, let’s take a look at this dump. With the dump file open in Wireshark, go to View→Name Resolution and make sure “Enable for Network Layer” is checked. This will improve readability by translating IP addresses to hostnames. The initial view (Figure 1) can be sort of intimidating, but there are some simple tips to make decoding this data easier.

As shown in Figure 1, Wireshark’s dump screen has one row per TCP packet, but the data is more easily



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	72.21.91.19	10.0.0.20	TCP	54	http > 51173 [RST] Seq=1 Win=0 Len=0
2	0.541909000	10.0.0.20	lga15s45-in-f19.1e100.	TLSv1	647	Application Data
3	0.595704000	lga15s45-in-f19.1e100.	10.0.0.20	TLSv1	616	Application Data
4	0.597877000	10.0.0.20	lga15s45-in-f19.1e100.	TCP	66	43947 > https [ACK] Seq=582 Ack=551 Win=576 Len=0 TSval=6006191
5	1.018033000	10.0.0.20	phx1-rb-api-wax-web-lt	TCP	54	47402 > http [FIN, ACK] Seq=1 Ack=1 Win=64240 Len=0
6	2.104025000	10.0.0.20	a72-247-9-209.deploy.a	TCP	66	53533 > http [FIN, ACK] Seq=1 Ack=1 Win=410 Len=0 TSval=6006382
7	2.105460000	10.0.0.20	a72-247-9-234.deploy.a	TCP	66	54436 > http [FIN, ACK] Seq=1 Ack=1 Win=364 Len=0 TSval=6006383
8	2.122524000	a72-247-9-209.deploy.a	10.0.0.20	TCP	66	http > 53533 [FIN, ACK] Seq=1 Ack=2 Win=8312 Len=0 TSval=104753
9	2.123835000	a72-247-9-234.deploy.a	10.0.0.20	TCP	66	http > 54436 [FIN, ACK] Seq=1 Ack=2 Win=8312 Len=0 TSval=131255
10	2.125950000	10.0.0.20	a72-247-9-209.deploy.a	TCP	66	53533 > http [ACK] Seq=2 Ack=2 Win=410 Len=0 TSval=6006386 TSec
11	2.126760000	10.0.0.20	a72-247-9-234.deploy.a	TCP	66	54436 > http [ACK] Seq=2 Ack=2 Win=364 Len=0 TSval=6006386 TSec
12	3.338431000	10.0.0.20	phx1-rb-api-wax-web-lt	TCP	54	[TCP Retransmission] 47402 > http [FIN, ACK] Seq=1 Ack=1 Win=64
13	3.466226000	10.0.0.20	10.0.0.1	DNS	80	Standard query 0xe6b2 A www.linuxjournal.com
14	3.537166000	10.0.0.20	10.0.0.20	DNS	96	Standard query response 0xe6b2 A 76.74.252.198
15	3.540379000	10.0.0.20	www.linuxjournal.com	TCP	74	33764 > http [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 T
16	3.599354000	www.linuxjournal.com	10.0.0.20	TCP	74	http > 33764 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 SAC
17	3.601774000	10.0.0.20	www.linuxjournal.com	TCP	66	33764 > http [ACK] Seq=1 Ack=1 Win=14656 Len=0 TSval=6006575 TS
18	3.602722000	10.0.0.20	www.linuxjournal.com	HTTP	517	GET / HTTP/1.1
19	3.662496000	www.linuxjournal.com	10.0.0.20	TCP	66	http > 33764 [ACK] Seq=1 Ack=452 Win=6912 Len=0 TSval=212947741
20	3.795165000	www.linuxjournal.com	10.0.0.20	HTTP	824	HTTP/1.1 302 Found (text/html)
21	3.797731000	10.0.0.20	www.linuxjournal.com	TCP	66	33764 > http [ACK] Seq=452 Ack=759 Win=17536 Len=0 TSval=600660
22	3.883485000	10.0.0.20	10.0.0.1	DNS	78	Standard query 0x4bbd A m.linuxjournal.com
23	3.956412000	10.0.0.1	10.0.0.20	DNS	94	Standard query response 0x4bbd A 76.74.252.198
24	3.959345000	10.0.0.20	www.linuxjournal.com	TCP	74	33765 > http [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 T
25	4.017206000	www.linuxjournal.com	10.0.0.20	TCP	74	http > 33765 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 SAC
26	4.019754000	10.0.0.20	www.linuxjournal.com	TCP	66	33765 > http [ACK] Seq=1 Ack=1 Win=14656 Len=0 TSval=6006629 TS
27	4.020944000	10.0.0.20	www.linuxjournal.com	HTTP	594	GET / HTTP/1.1

▶ Frame 16: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
 ▶ Ethernet II, Src: IntelCor_28:5b:cc (a4:4e:31:28:5b:cc), Dst: SamsungE_e7:c9:fb (a0:0b:ba:e7:c9:fb)
 ▼ Internet Protocol Version 4, Src: www.linuxjournal.com (76.74.252.198), Dst: 10.0.0.20 (10.0.0.20)
 Version: 4
 Header length: 20 bytes
 ▶ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
 Total Length: 60
 Identification: 0x0000 (0)
 ▶ Flags: 0x02 (Don't Fragment)
 Fragment offset: 0

Figure 1. Wireshark Output

consumed when reassembled into a full TCP stream. To get the full stream, right-click on any row where the source or destination is `www.linuxjournal.com`, and choose “Follow TCP Stream”. This automatically will find all the related packets and group them together in an easier-to-read format.

In this example, you can see the HTTP GET request from my phone in red, and the HTTP response from the *Linux Journal* Web server in

blue. Here is where you can start to see unencrypted information flowing back and forth from the server. Since the server response’s “Content-Type” header indicates that the response is a JPEG image, you can view that image with a little bit of extra manipulation. Press the “Save As” button to save the stream to a temporary file (use RAW format), then use an editor like `emacs` or `vi` to trim out the header text from the image

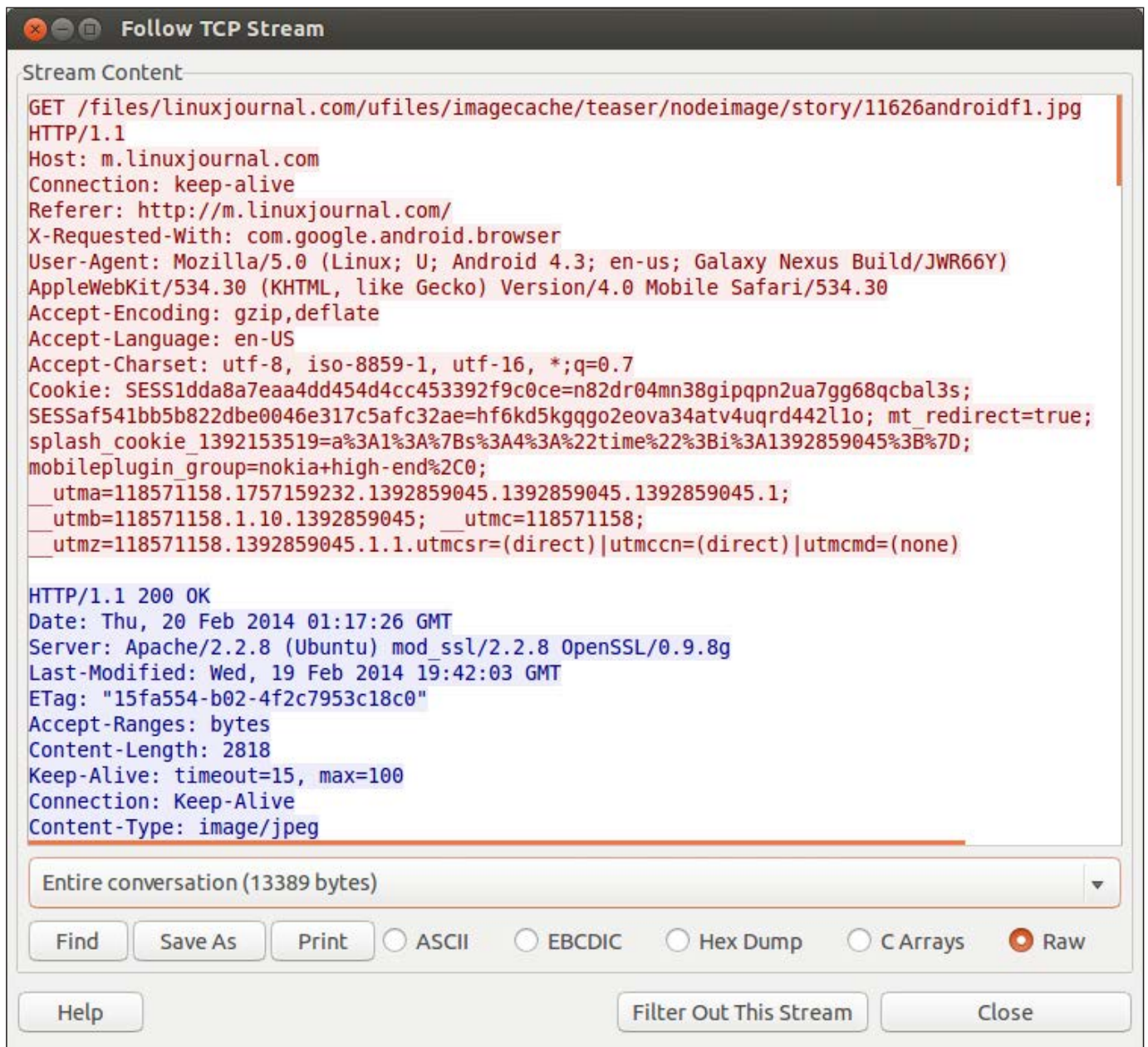


Figure 2. Follow TCP Stream

binary contents. It takes a little bit of practice, but it's usually pretty obvious where the HTTP header stops and the binary bits begin.

Once you've removed the header (and any stray footer or additional header sections), you can save the file

with a .jpeg extension and view it.

Continue browsing through the dump manually and look for interesting TCP segments. You also could take a more systematic approach by using Wireshark's filtering capabilities. Use a filter like

[illegible]

Figure 3. Raw TCP Dump

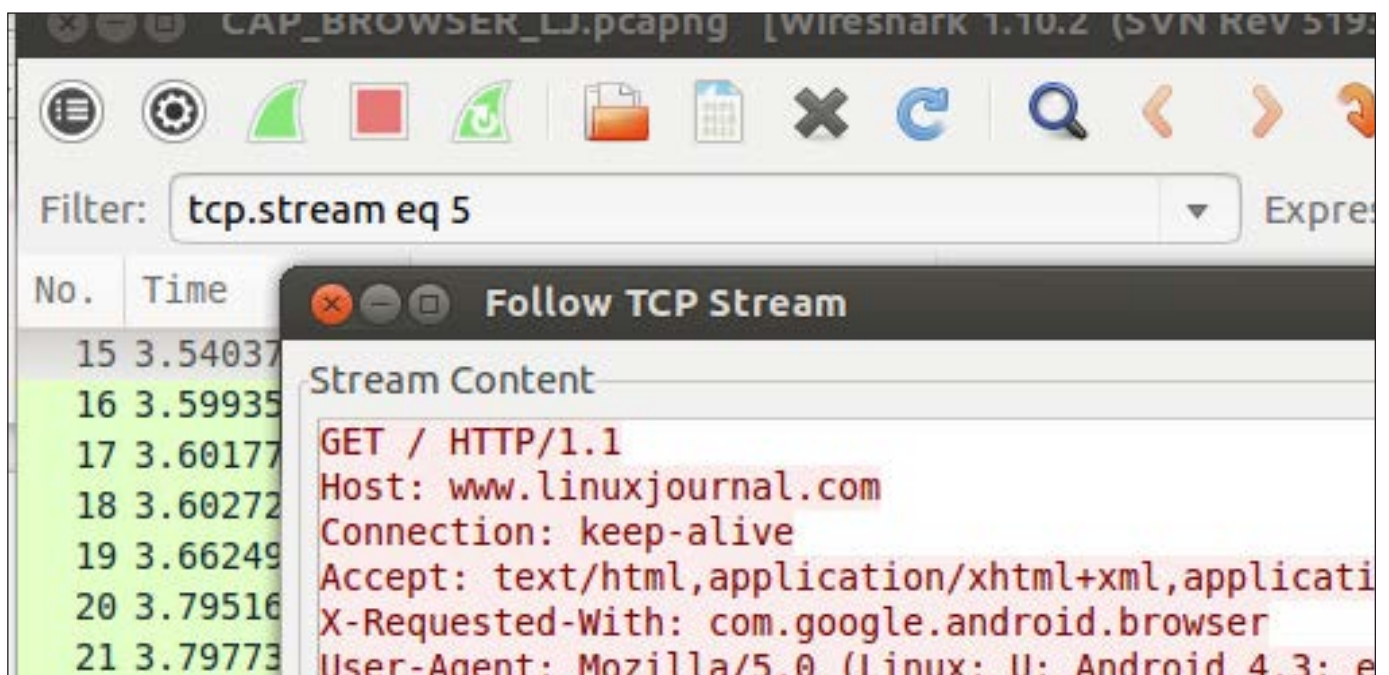


Figure 4. Filtering to a Single TCP Stream

tcp.stream eq 1 (Figure 4), and keep iterating the stream ID until you've seen all the streams, drilling down with "Follow Stream" if the packets look promising.

Capturing Low-Sensitivity Application Traffic

Now that you're getting a little more comfortable with capturing and viewing dumps with Wireshark, let's try peeking at the information coming to and from an Android application. For this next test, I used the app "reddit is fun" since it sends and receives non-sensitive data that is probably not encrypted.

Capture an app search or query using the same technique as before: start Wireshark on the laptop, launch

and exercise the app from the phone, then stop Wireshark and save the capture file.

Figure 5 shows an example TCP stream from "reddit is fun".

Again, the request from the app is in red, and the response from the reddit server is in blue. Note that since the request is not encoded, anyone monitoring the WAP would be able to detect your interest in "Raspberry Pi" data. The content-type of the response is JSON, and even though the Content-Encoding is set to "gzip", Wireshark is letting you view the content body as pure JSON. If the data in your TCP Stream page looks garbled, you may have an older version of Wireshark that doesn't support on-the-fly gzip decoding.

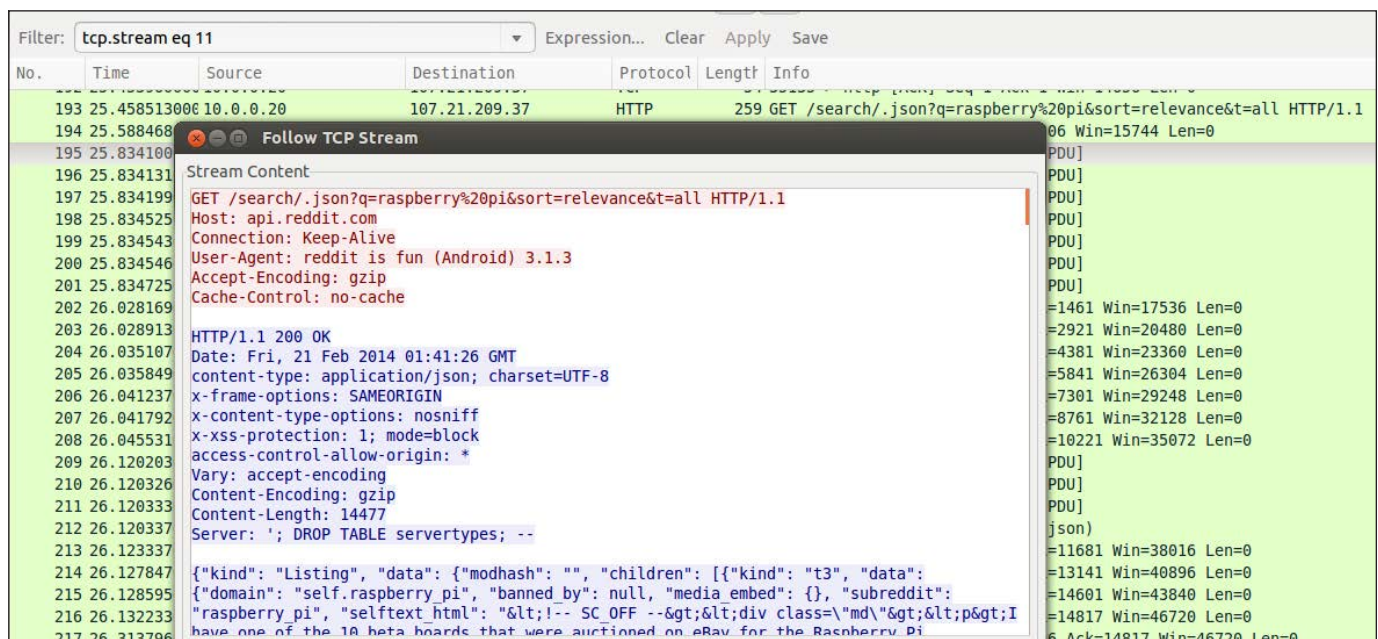


Figure 5. Gzip-Encoded JSON

Either save the contents to a file and gunzip on your own, or upgrade your version of Wireshark.

Note: look at that hilarious “Server” header in the response—is some clever reddit engineer sending an SQL injection attack to some script kiddies?

Capturing High-Sensitivity App Data

By now, the process to capture traffic from an app should be

pretty straightforward. Let’s try running a banking or high-sensitivity app and use the tricks described earlier to see if you can detect the application sending any information in the clear that it shouldn’t. To be perfectly honest, the odds of finding such a low-level (and easily avoidable) flaw are going to be very, very low. Android application development is pretty mature now, and the

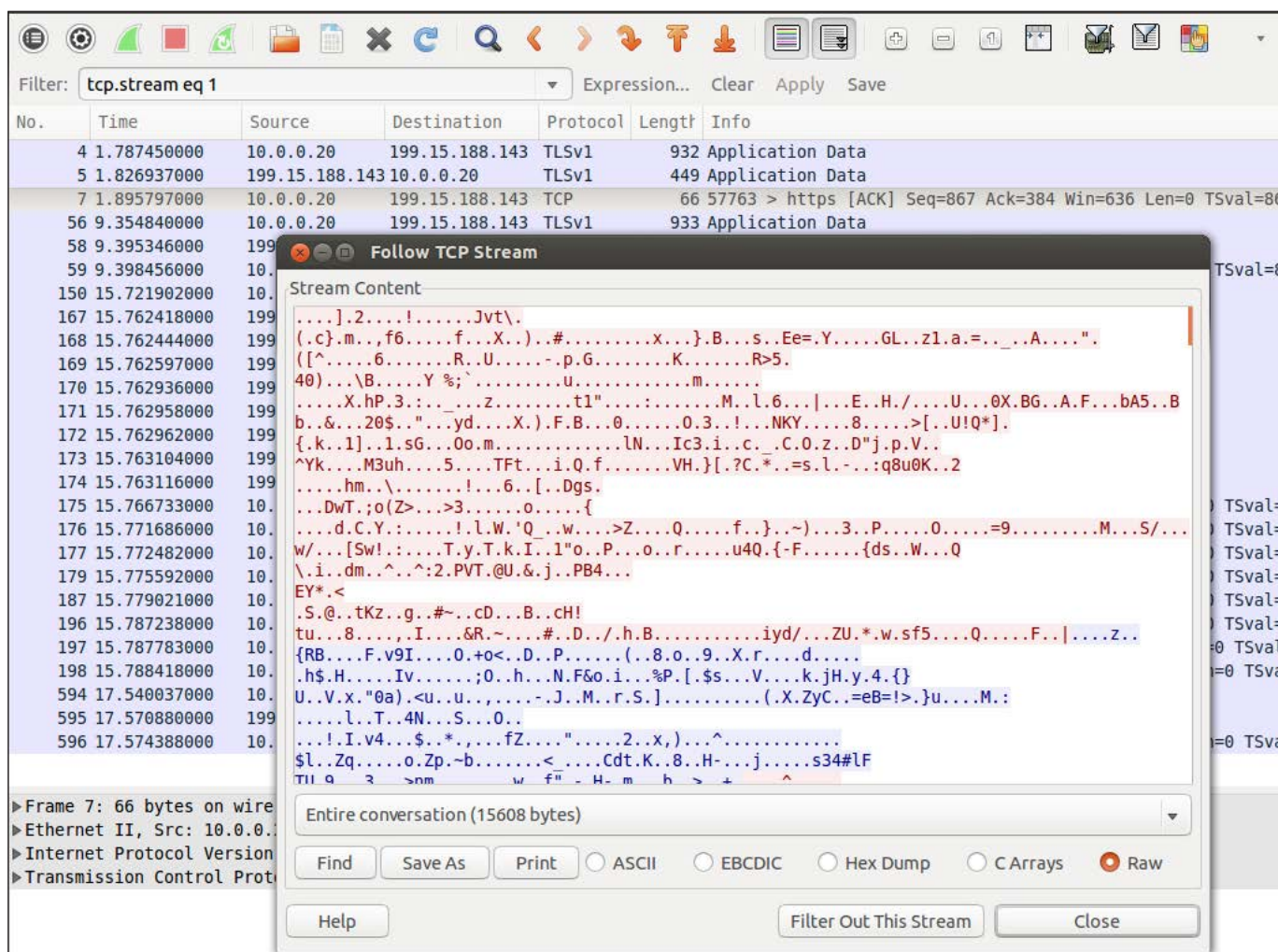


Figure 6. Encrypted Traffic

Android libraries make using SSL encryption pretty easy. It feels good to double-check though, so follow the same steps as before, but log on to a banking application of your choice.

Now, as you step through the TCP streams, you should note a few major differences. Most of the traffic will be HTTPS instead of HTTP, and the protocol will be TLS instead of TCP or HTTP. In addition, the TCP stream no longer will contain human-readable content, even after trying the standard gunzip tricks (Figure 6).

Step through the TCP streams, following each one, and verify that there's no plain text or unencrypted communications that are exposing anything scary.

Next Steps

Now that you've almost certainly *not* found anything scary, where else can these network monitoring skills be applied? Here are some fun ideas:

- Attach a console like a Wii or PS3 and see what kind of information it sends at startup and login.
- Create a WAP that doesn't actually go anywhere and

just see what tries to connect. Maybe there's a device using Wi-Fi that you didn't even know about?

- Get the SSL certificate for a server you support, and try out Wireshark's SSL decoding.
- Reverse the wlan0 and eth0 designations in the scripts and set up the system backwards (connect the laptop's Wi-Fi to your existing WAP, and plug a device in to the laptop's Ethernet port) to monitor the output of wired-only devices. My "smart" Blu-ray player was communicating with all sorts of unexpected places at startup! ■

Brian Trapp serves up a spicy gumbo of Web-based yield reporting and analysis tools for hungry semiconductor engineers at one of the leading semiconductor research and development consortiums. His signature dish has a Java base with a dash of JavaScript, Perl, Bash and R, and his kitchen has been powered by Linux ever since 1998. He works from home in Buffalo, New York, which is a shame only because that doesn't really fit the whole chef metaphor.

|||||

Send comments or feedback via
<http://www.linuxjournal.com/contact>
or to ljeditor@linuxjournal.com.

OSSIM

a Careful, Free and
Always Available
Guardian for
Your Network

Monitor your network's security 24/7 with a free and open-source solution that collects, analyzes and reports logs of the events on your network.

MARCO ALAMANNI

Networks and information systems are increasingly exposed to attacks that are becoming more sophisticated and sustained over time, such as the so-called APT (Advanced Persistent Threats).

Information security experts agree on the fact that no organization, even the best equipped to protect itself from these attacks, can be considered immune, and that the issue is not whether its systems will be compromised, but rather when and how it will happen.

It is essential to be able to detect attacks in a timely manner and implement the relative countermeasures, following appropriate procedures to respond to incidents, thus minimizing the effects and the damages they can cause. In order to detect intrusions and attacks, system administrators and information security analysts make use of tools, such as IDS/IPS (Intrusion Detection/Prevention System) and analysis of logs (event records) of servers and network devices, looking for any significant events from a security point of view.

A network of an organization of average size produces, as a whole, such a quantity of logs that it is very difficult (and still very expensive) to

check them all, one by one, to obtain meaningful information.

A further difficulty is that there is no single standard used to record the logs and often, depending on the type and size, they are not immediate or easy to understand.

It is even more difficult to relate other logs produced by many different systems to each other manually, to highlight anomalies in the network that would not be detectable by analyzing the logs of each machine separately.

SIEM (Security Information and Event Management) software, therefore, is not limited to being a centralized solution for log management, but also (and especially) it has the ability to standardize logs in a single format, analyze the recorded events, highlight the most important information and relate the logs to each other (correlation), allowing analysts to detect anomalies and attacks more easily.

For example, for log management software, three failed attempts to log in to the same user account from three different clients will be only three lines in your log file and not obviously related to each other. For an analyst, instead, it may be a sequence of events worthy of further analysis, and

its correlation (looking for patterns in the log files) can generate alerts when these types of events occur.

Overview of a SIEM Open-Source Solution: OSSIM

OSSIM is a SIEM software platform, free and open-source, developed by AlienVault and based on a Debian 64-bit Linux distribution. OSSIM has four major components:

1. Sensor.
2. Server.
3. Framework.
4. Database.

You can install these components on a single physical machine (the default installation), on a single virtual machine, on different virtual machines and/or physical machines, depending on the size and configuration of the network to monitor.

For a relatively small network, installation on a single machine, which is the simplest configuration, may be the right solution. For larger networks, it is advisable to install the Sensor and the Database separately. Figure 1 shows the OSSIM architecture.

Sensor: The Sensor has two main components:

1. The rsyslog service, which listens

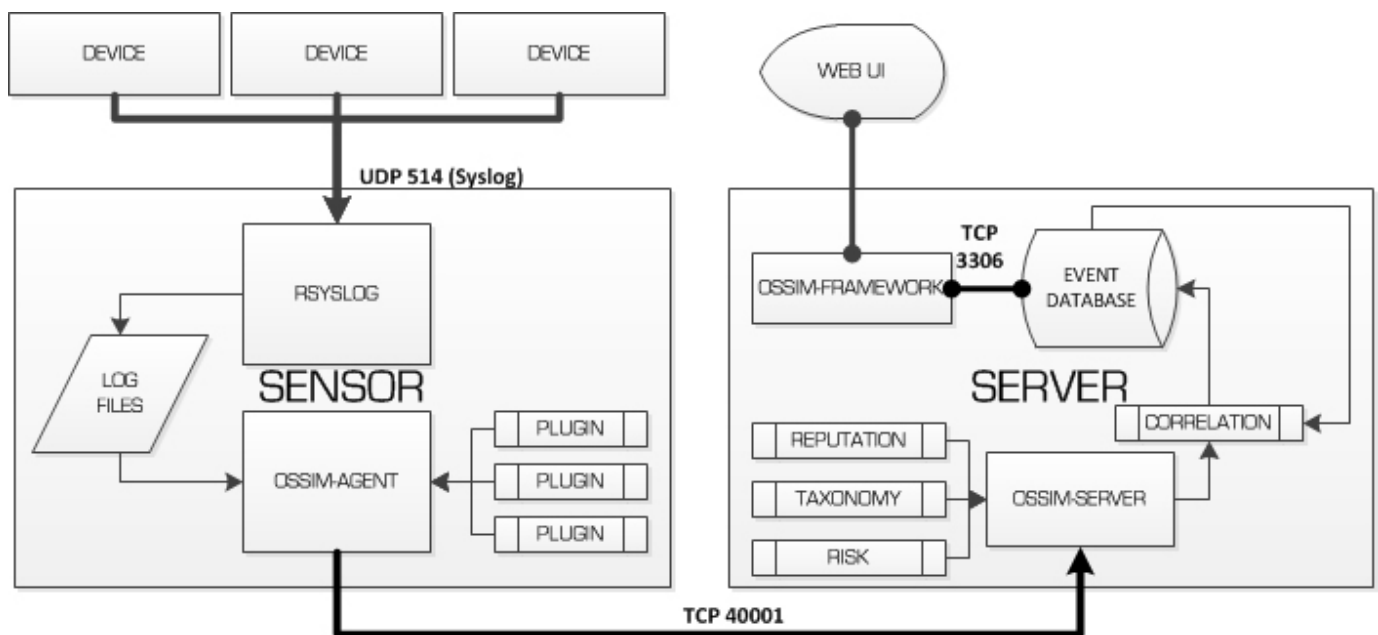


Figure 1. OSSIM Architecture

on TCP/UDP port 514, receives the logs from network devices and stores them locally, according to the configuration.

2. The Ossim-agent, using a series of modules called plugins, one for each type of log, performs log analysis and normalization, and sends that to the Server component.

Plugins are of two types: detectors, which detect anomalies and possible attacks (such as Snort, P0f, Arpwatch), and monitors to monitor the network status (like Ntop and Nagios).

Server: The Server performs the essential SIEM functions: aggregation, risk assessment and correlation of events that are received from the sensor through TCP port 40001. The server also sends the information concerning the events to the Database for storage.

Framework: The Framework connects and manages the OSSIM components and security tools included, and it provides the system administration Web interface. It is the component that needs the least hardware resources and is usually installed together with the Server component.

Database: The Database is a MySQL server instance that stores events and system configuration data.

Functionalities

Following is a brief description of OSSIM's main features and functionalities concerning the collection, analysis and correlation of logs and the primary tools included in the system for network security monitoring.

Collection and Normalization of Logs: You can collect logs from the devices on your network in two ways:

1. Install a software agent (like Snare or SysLogAgent) in the source machine and configure it to read certain types of logs and send them to the Sensor component.
2. Configure the source machine to send the logs upon request of the appropriate Sensor plugins (for example, via WMI for Windows machines). Once the Sensor records the logs, the OSSIM Agent performs the analysis and converts them to a single format (normalization). Each log represents an event that will be sent to the server for analysis (Figure 2).

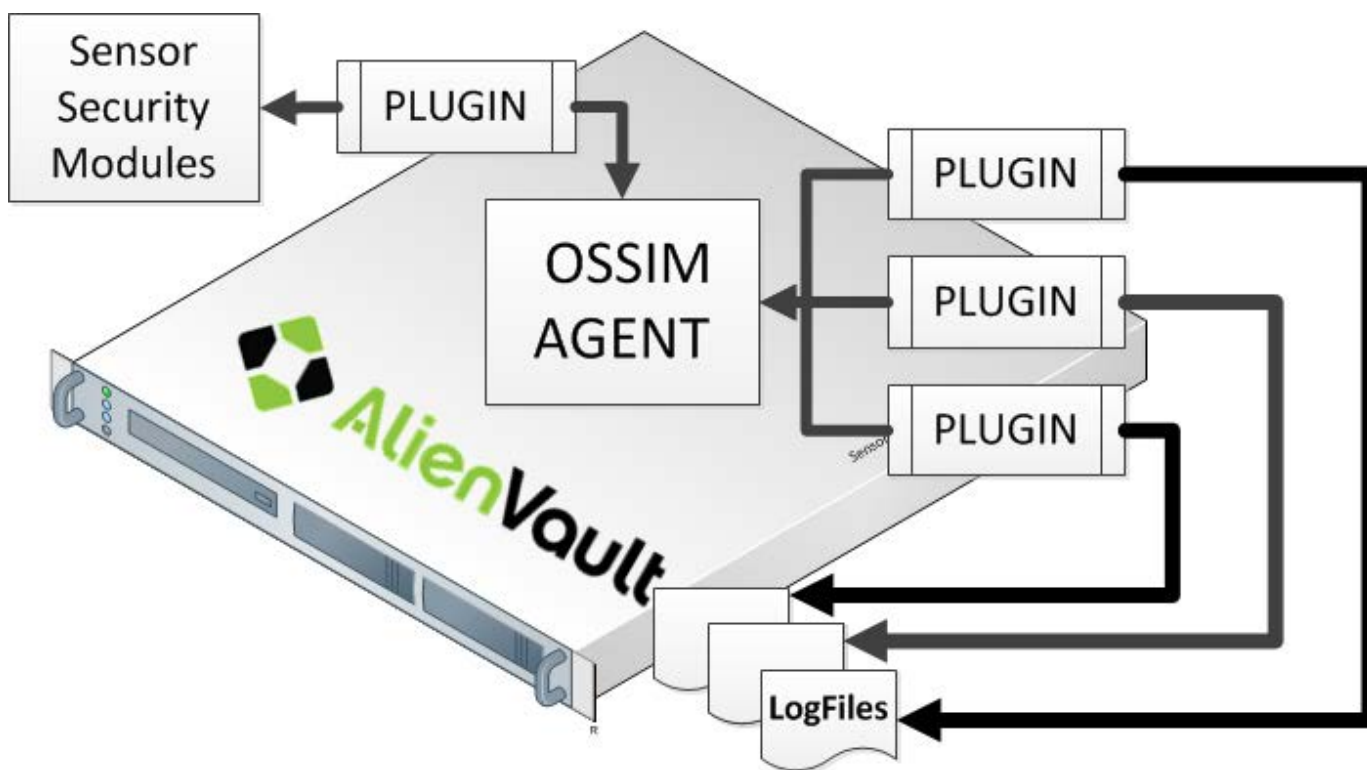


Figure 2. Log Collection and Normalization

Prioritization of Events and Risk Assessment:

The prioritization process involves assigning priority values to the recorded events, which is done by the Server component. It depends on the structure of the network and its needs, as prerequisites, the definition of security policies and the inventory of information assets on the network, which can be managed in the Web administration panel. It sets the priority of an event based on the machine that generated it and the type of event to which it belongs.

The risk assessment of events is

calculated in real time and is based on three main factors:

1. The value or level of importance of the machine that generated the event.
2. The type of threat posed by the event.
3. The probability that this event occurs.

The formula used for calculating risk is the following (Figure 3):

$$\text{Risk} = \text{value} * (\text{reliability} * \text{Priority} / 25).$$

Analysis and Correlation of Events: The correlation of events essentially relates events to each other

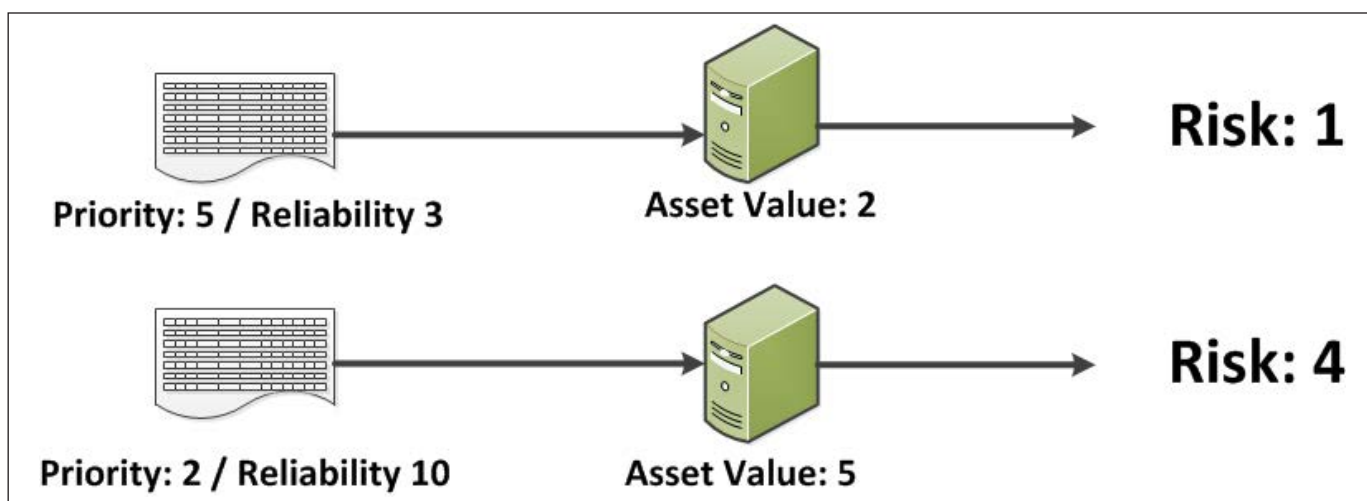


Figure 3. How to Calculate the Risk Associated with an Event

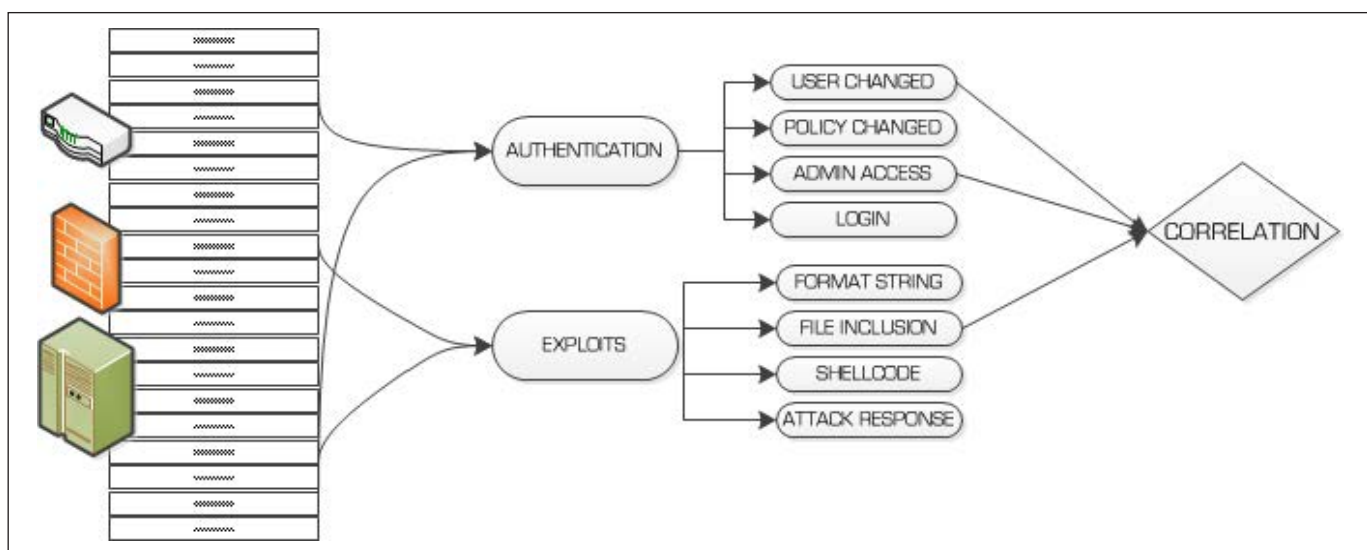


Figure 4. Example of Analysis and Correlation of Events

to achieve a comprehensive view of network security and to detect possible attacks or anomalies.

The correlation process is performed via two methods:

1. Correlation using sequence of

events, using directives, consisting of rules that relate events to patterns of known attacks. This method is similar to using Snort for intrusion detection (signature-based detection).

2. Correlation using heuristic

algorithms can be detected by these abnormal situations that do not detect the preceding rules and may or may not be attacks (abnormality detection).

Directives are located in the `/etc/ossim/server/directives.xml` file. Directives are specified in XML using tags like Id, Name, Priority, Type, Reliability, Occurrence, Timeout, Source, Destination, Source port, destination port, protocol, PluginSid and Sensor.

Reliability is a measure of the probability that the considered event truly represents the attack referred to by the directive and is generally based on the number of occurrences of the event.

For example, consider the following directive to detect brute-force SSH attacks:

```
<directive id="20" name="Possible SSH brute force login"
  ➤attempt against DST_IP" priority="5">
<rule type="detector" name="SSH Authentication failure"
  ➤reliability="3" occurrence="1" from="ANY" to="ANY"
  ➤port_from="ANY" port_to="ANY" time_out="10"
  ➤plugin_id="4003" plugin_sid="1,2,3,4,5,6">
<rules>
<rule type="detector" name="SSH Authentication failure (3 times)"
  ➤reliability="+1" occurrence="3" from="1:SRC_IP" to="ANY"
  ➤port_from="ANY" time_out="15" port_to="ANY"
  ➤plugin_id="4003" plugin_sid="1,2,3,4,5,6" sticky="true">
```

```
<rules>
<rule type="detector" name="SSH Authentication failure (5 times)"
  ➤reliability="+2" occurrence="5" from="1:SRC_IP" to="ANY"
  ➤port_from="ANY" time_out="20" port_to="ANY"
  ➤plugin_id="4003" plugin_sid="1,2,3,4,5,6" sticky="true">
<rules>
<rule type="detector" name="SSH Authentication failure (10 times)"
  ➤reliability="+2" occurrence="10" from="1:SRC_IP" to="ANY"
  ➤port_from="ANY" time_out="30" port_to="ANY"
  ➤plugin_id="4003" plugin_sid="1,2,3,4,5,6" sticky="true">
</rule>
</rules>
</rule>
</rules>
</rule>
</rules>
</rule>
</rules>
</directive>
```

The directive assigns a value of reliability equal to 3 (30% probability) when the number of occurrences of the event detected by the sensor (SSH authentication error) is equal to 1, then increments it by 1 at the third occurrence of the event, by 2 at the fifth occurrence and by an additional 2 at the tenth, thereby achieving a reliability of 8 (80% of probability) when the incorrect authentication attempts are 10.

OSSIM also has the ability to correlate different types of logs, generated by various plugins (cross-correlation). The cross-correlation

allows you to change the event reliability and risk assessment. For example, suppose that Nessus or OpenVAS has identified a vulnerability in a server. If Snort detects an event that indicates a possible attack on that server, the correlation engine increases the level of risk associated with the event.

Generation of Alarms and Response Actions: The directives can create alarms, which either are generated by a single event or by a specific sequence of events under certain conditions. The alarms can be displayed in the Web administration panel, under the menu item Incidents→Alarms.

Furthermore, alarms can activate response actions, such as sending an alert by e-mail to the system administrator and/or the execution of appropriate scripts.

Vulnerability Analysis, Intrusion Detection and Network Monitoring: OSSIM includes many valuable tools, which also are open-source, that are among the most known and used for intrusion detection, vulnerability analysis and network management and monitoring:

- Arpwatch: used for monitoring ARP traffic on the LAN and for related attack detection.

- P0f: used for operating system identification and analysis.
- Pads: used for detecting anomalies of the services running on a host.
- Nessus and OpenVAS: the most widely used and popular vulnerability scanners.
- Nmap: the most famous and powerful network scanner.
- Snort: the most popular intrusion detection system (IDS).
- Tcptrack: used for TCP connection monitoring.
- Nagios and Ntop: used to monitor the status of the network, the hosts and the availability of services.
- Osiris and OSSEC: intrusion detection software for individual hosts (HIDS—Host-Based IDS).
- Snare: a software agent for collecting logs on Windows systems.

Installation and Hardware Requirements

You can download the ISO file for the installation from the

AlienVault Web site download page at <http://www.alienvault.com/free-downloads-services>.

The most recent version (February 2014) is 4.3.4, only for 64-bit architectures. You can choose the Automatic or Custom installation. The automatic installation is fairly simple, in graphical mode by default, and it installs all components of OSSIM on the same machine. The custom installation allows you to select the mode (graphical or textual) and which components to install. The custom installation is a little more complex because it has more configuration options. For instructions on how to install OSSIM, refer to the Installation Guide: <https://alienvault.bloomfire.com/posts/525575-installation-guide/public>.

The minimum hardware requirements are:

- 64-bit processor or virtualization software with support for 64-bit operating systems (at least a quad-core processor is recommended).
- 4GB of RAM.
- 500GB of free disk space.
- Network adapter with support for

the Intel e1000 Ethernet driver.

Of course, the hardware requirements will be directly proportional to the size of the network (number of hosts and network devices connected) and consequently to the amount of logs produced and recorded.

Configuration and Management

You can perform the system configuration and administration through the console, a Linux shell or through a more convenient and intuitive Web interface.

Configuration through the Console: To configure the system through the console, you need to log in as root with the password you set during the installation process. The directory that contains the system's configuration files is `/etc/ossim`.

The main configuration file is `/etc/ossim/ossim_setup.conf`, which contains the system's main settings, such as IP addresses and ports of the hosts on which components are installed, the active plugins and the password used by the root user of MySQL, randomly generated by the system during the installation procedure.

For example, if you want to

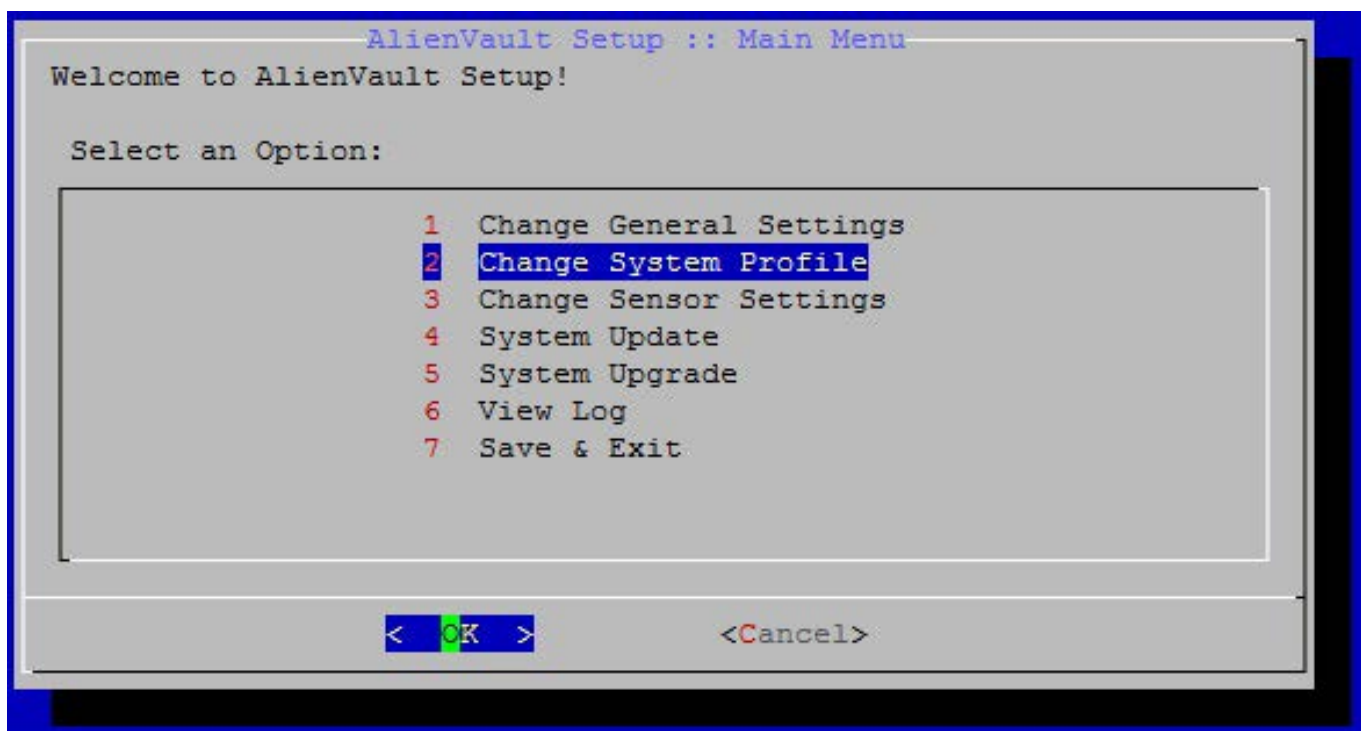


Figure 5. Configuration with the `ossim_setup` Tool

change your password or other data, you need to edit the file with the command:

```
# vi /etc/ossim/ossim_setup.conf
```

Then run the following command:

```
# ossim-reconfig
```

To change the main configuration file more easily, there is also a command called `ossim_setup`, which presents a graphical interface, shown in Figure 5.

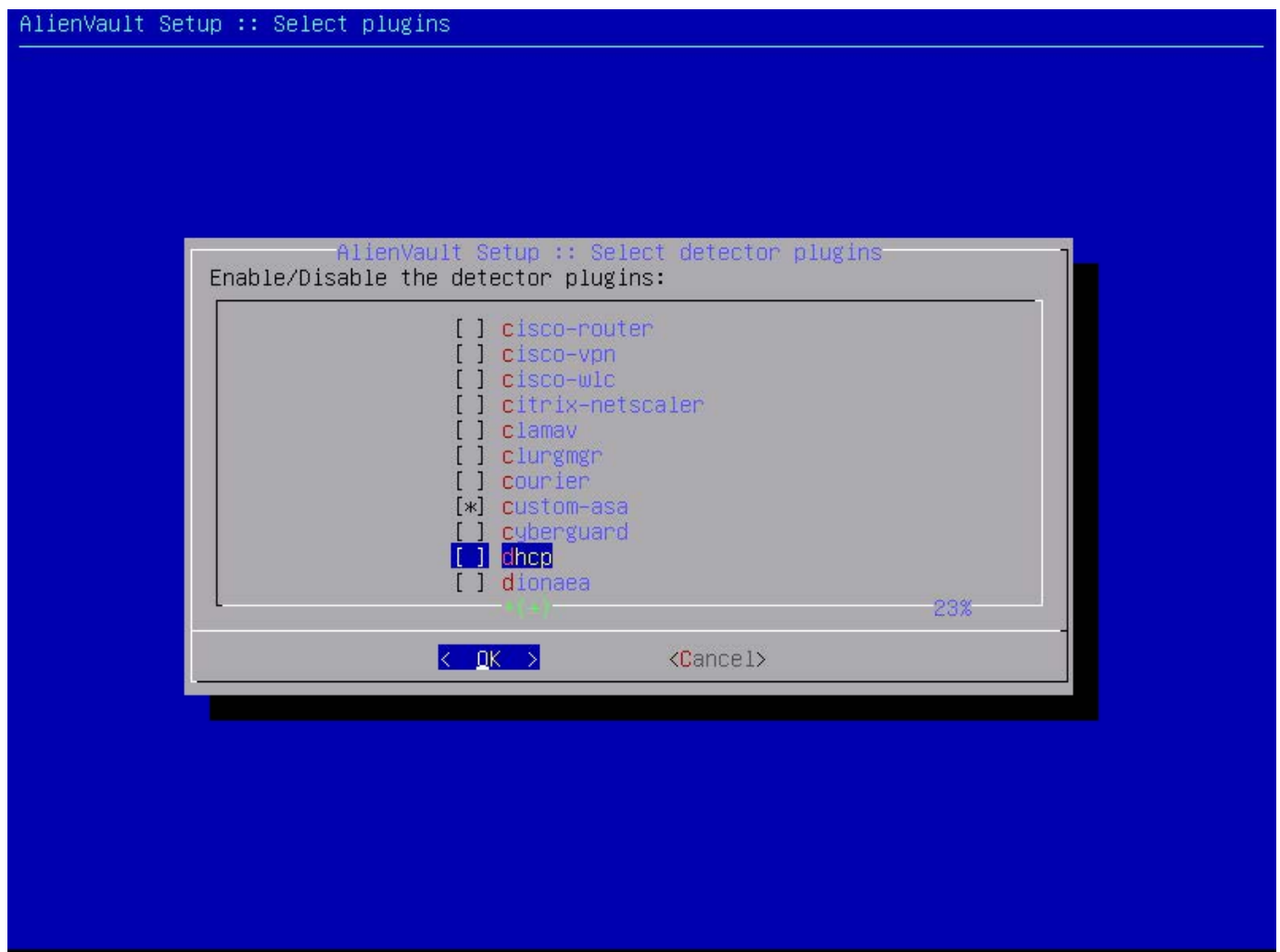
Configuring the Sensor and Plugins: With the `ossim_setup` command, you can set the

parameters of the previously shown configuration file, such as enabling or disabling plugins. To get a list of plugins that can be turned on or off, select the option Change sensor settings→Select detector plugins (Figure 6).

The OSSIM agent runs as a background service (dæmon), and you can start it with this command:

```
# /etc/init.d/ossim-agent start
```

Its configuration file is `/etc/ossim/agent/config.cfg`. The plugins' configuration files also are text files with the `.cfg` extension and are in the `/etc/ossim/agent/plugins/` directory.

**Figure 6. List of Plugins in ossim_setup**

When you activate new plugins, you must restart the server:

```
# /etc/init.d/ossim-server restart
```

More than 2,000 plugins are available (<http://www.alienvault.com/docs/AlienVault%20Plugin%20List%20-%20Jun-20-2010.pdf>), which you can download and install via the Plugin Wizard. For example, run the following commands:

```
# cd /usr/share/ossim/scripts ;  
# plugin_wizard.pl -s "oracle"
```

and you will get the plugins that contain the word "Oracle" in the name.

With the command:

```
# ./plugin_wizard.pl -g -s "oracle"
```

these plugins will be extracted to a directory called win_plugins.

Next, you have to move them

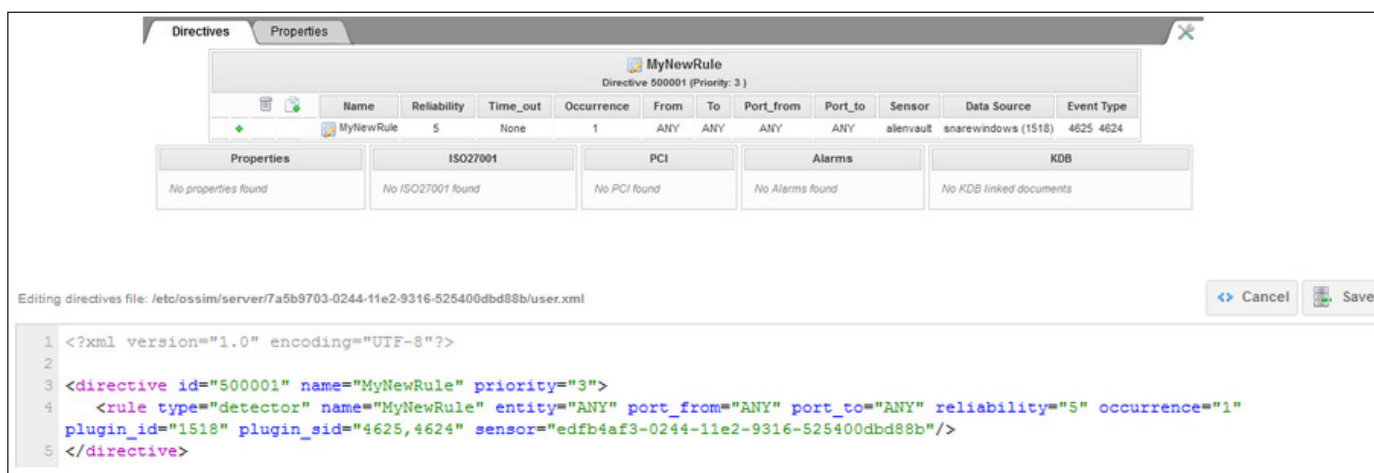


Figure 7. New Directives Specified in the user.xml File

to the default directory /etc/ossim/agent/plugins/:

```
# mv win_plugins/*.cfg /etc/ossim/agent/plugins/
```

Files with the .sql extension must be added to the MySQL database with the following command:

```
# ossim-db < ./win_plugins/*.sql
```

If the database server is not installed on the same machine, you need to copy the files on the server:

```
# scp win_plugins/*.sql root@<IP Database>:/root/
```

and run the ossim-db command from the database server.

Server Configuration: The configuration directory for the Server component is /etc/ossim/server. The main file is directives.xml,

which specifies the configuration file directives, grouped by type of attack, such as malware, brute force and so on.

When you create new guidelines, they should be specified in user.xml rather than in the file containing the default directives (Figure 7).

rsyslog's Dæmon Configuration and Log Rotation: The /var/log/ossim directory contains the log files of OSSIM's components. The dæmon that keeps track of the logs is, as already mentioned, rsyslog, whose configuration file is /etc/rsyslog.conf.

During the installation process, you configure rsyslog to accept logs from remote machines and store them in different log files, depending on the type and the host that created them.

To achieve this, rsyslog uses filters

based on expressions, which are .conf files usually placed in the /etc/rsyslog.d/ directory. For example, to save the logs from a Fortinet firewall in the file /var/log/ossim/fortinet.log, the expression would be:

```
if ($source == '192.168.1.100' and $msg contains 'fortinet ')
    and $severity <= '6' then /var/log/ossim/fortinet.log
```

Adding new hosts that send the logs to rsyslog, you quickly can run out of disk space. Therefore, it is important to define a policy for log rotation in /etc/logrotate.conf. This involves the regular archiving, at predefined intervals, of the existing log files. After a predefined period, the archived log files are deleted or stored on external devices for backup.

Administration through the Web Interface: OSSIM also can be configured and managed through a nice Web interface, connecting with the browser to the IP address of the machine on which you installed the Server/Framework component (Figure 8).

The default user and password are admin/admin. When you log in for the first time, you are prompted to change your password.

Through the Web interface, you can perform the following tasks:

- System configuration (users,



Figure 8. Web Administration Interface Login Screen

update, backups and so on).

- Creation and configuration of directives, policies and actions.
- Real-time monitoring of network security.
- Report generation.
- Ticketing system.
- Vulnerability management and incident response.
- Management and optimization of network traffic.

The Web interface includes

several sections:

- **Dashboard:** provides an overview of detected security events. Displays the visual counters and statistics of the most important security events (Figure 9).
- **Incidents:** shows the list of security events and generated alarms with specific information, such as date, priority, risk, status

and a brief history of the actions taken by system administrators.

- **Analysis:** shows a table with the latest events detected, the type, date, origin, destination, the OSSIM node that detected it and the risk. From here, the user can search for patterns in the events according to different criteria (for example, the source IP address). This includes a real-time list of

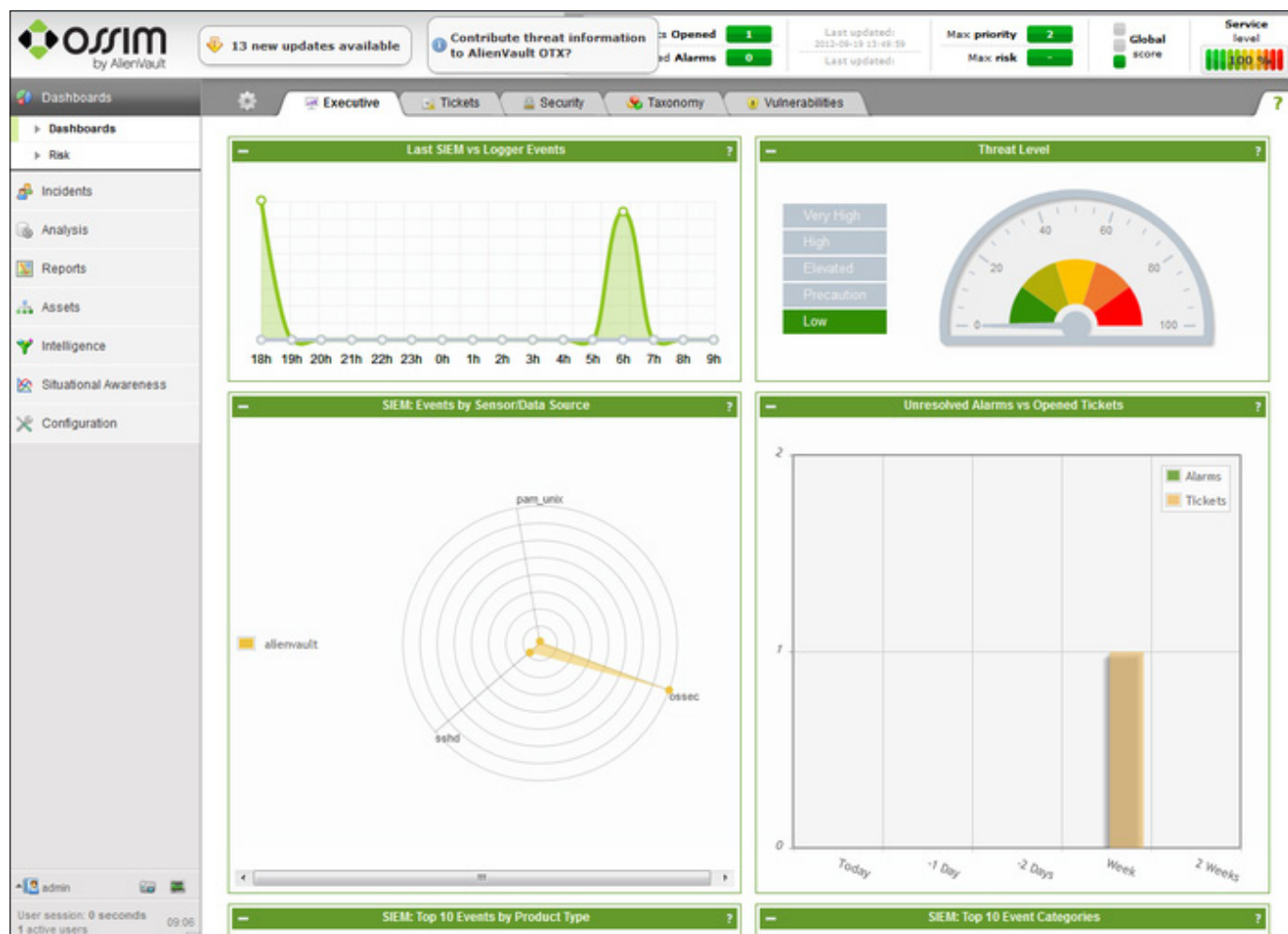


Figure 9. Dashboard with Statistics and Diagrams about Security Events

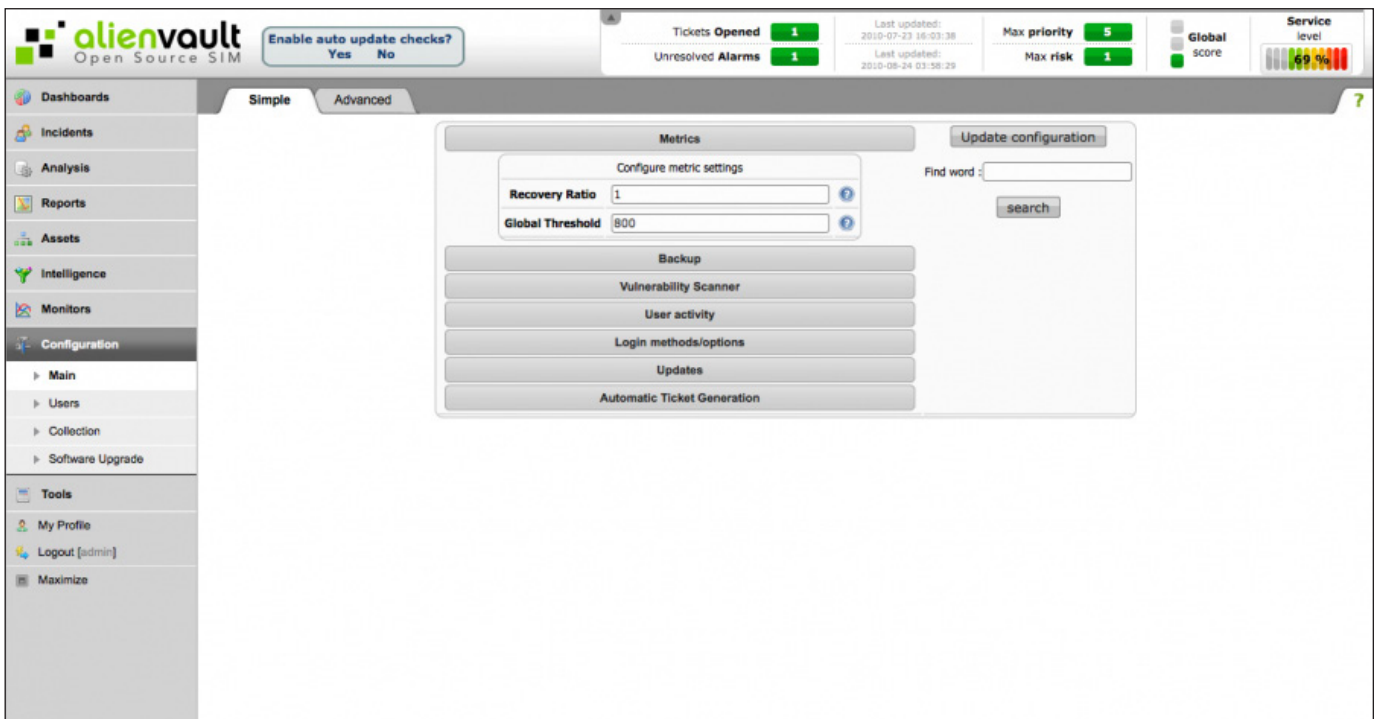


Figure 10. This section manages the system logic: definition and management of policies, directives and actions.

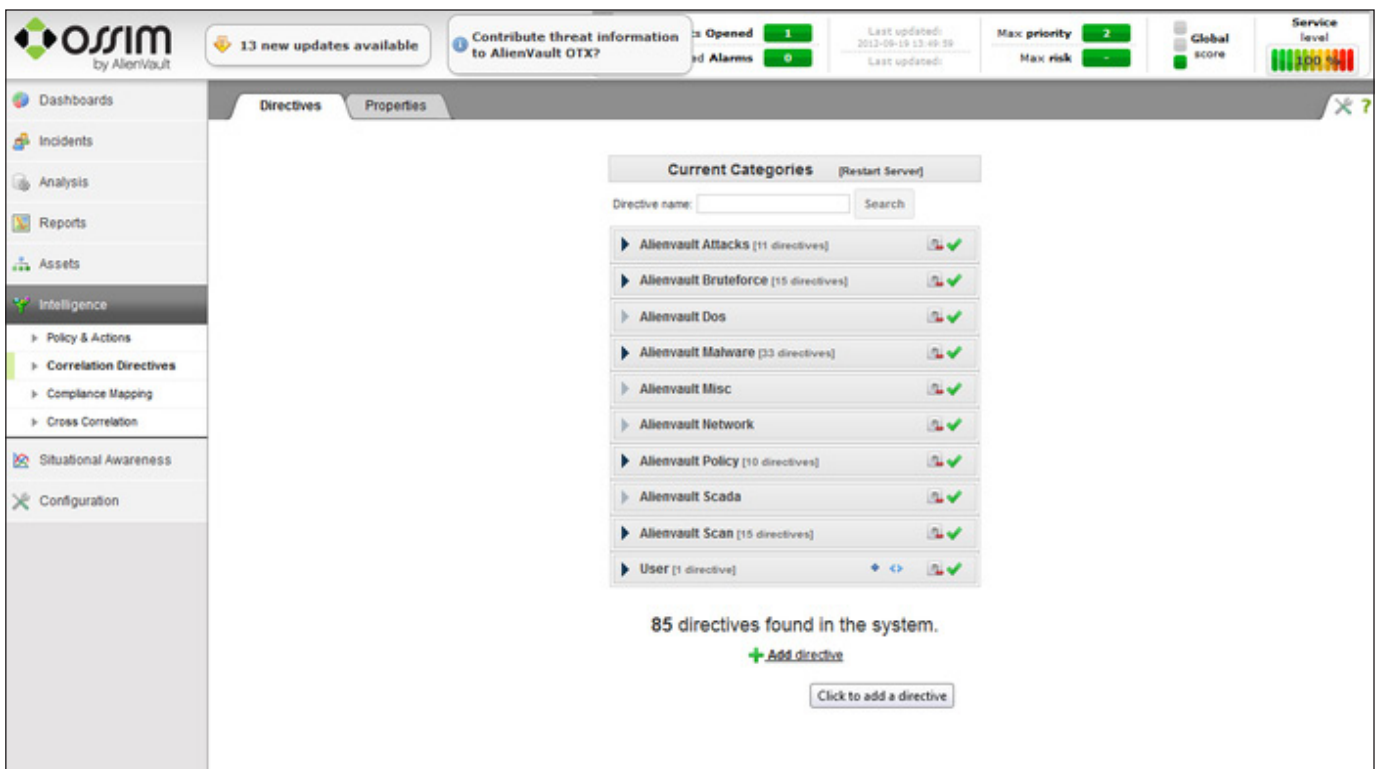


Figure 11. Configuration Panel

detected events that is updated every two seconds.

- **Report:** allows you to generate reports about security events and network status.
- **Activities:** this interface allows you to run and manage network inventory, identify and add new machines from which to record the logs.
- **Intelligence:** this section handles the system logic—definition and management of policies/actions, directives, event correlation and statistics of the network and of the OSSIM nodes.
- **Configuration:** this section allows you to manage all the system configurations (Figure 10).

Conclusion

OSSIM is a viable open-source SIEM

solution and a free alternative to other commercial SIEM products (including AlienVault USM, the commercial version of OSSIM), which are much more expensive, and it is supported by a community of developers and users through forums and documentation available on the AlienVault's Web site. ■

Marco Alamanni has professional experience working as a Linux system administrator and information security administrator in banks and financial institutions in Italy and Peru. He holds a BSc in Computer Science and an MSc in Information Security, and his interests in information technology include ethical hacking, digital forensics, malware analysis, Linux and programming. He also collaborates with IT magazines writing articles about Linux and IT security.

Send comments or feedback via
<http://www.linuxjournal.com/contact>
or to ljeditor@linuxjournal.com.

Resources

OSSIM Installation Guide: <https://alienvault.bloomfire.com/posts/525575-installation-guide/public>

AlienVault User Manual: http://www.alienvault.com/wiki/doku.php?id=user_manual:introduction

The Alienvault Repository of Knowledge: <https://alienvault.bloomfire.com>

AlienVault OSSIM Forum: <http://forums.alienvault.com>

Service Level SIEM—User and Programmer Guide: [http://forge.fi-ware.org/plugins/mediawiki/wiki/fiware/index.php/Security_Monitoring /Service_Level_SIEM - User and Programmers Guide](http://forge.fi-ware.org/plugins/mediawiki/wiki/fiware/index.php/Security_Monitoring /Service_Level_SIEM_-_User_and_Programmers_Guide)

Berkeley Packet Filters with Scapy (and Friends)

Get to know the language and tools that can take your network filtering rules to a whole new level.

VALENTINE SINITSYN

Network filtering is probably as old as networks themselves. If you exchange data with the outside world, it's natural to control what's going in and out. If, however, you capture packets for a good reason (like traffic analysis or intrusion detection), filtering those you are not interested in as early as possible is crucial for performance. The first type of filtering is typically done with a firewall. Berkeley Packet Filter (BPF) is what comes to the rescue in the second case.

Originally, BPF referred to both the capturing technology and its high-performance filtering capabilities. For some Unices (for instance, FreeBSD), this still holds true, and there is a `/dev/bpf` device from which you can read captured packets. For others, BPF means what it says (the filter), and in this reincarnation, you can find it in various operating systems, including Linux, and with WinPcap and even Microsoft Windows.

BPF filters are programs written in a low-level language similar to assembler (I take a look at that in the last section of this article). These programs are executed by a BPF virtual machine. Most often, this virtual machine resides in the kernel and uses Just-In-Time compilation (JIT) to boost filtering performance.

However, user-mode BPF interpreters (useful for debugging purposes or as fallbacks) are available as well—for instance, the one provided with the ubiquitous libpcap library (<http://www.tcpdump.org>), the main workhorse behind tcpdump, Wireshark and other popular network tools.

All of this may sound like you need to delve into registers and opcodes just to say you are interested in the packets coming to host 192.168.1.1. Luckily, that's not the case. Of course, if you want (or need) to, you can, but libpcap provides its own high-level filter language that compiles directly to BPF. Largely, this syntax is synonymous with BPF (they are tightly related albeit different things), so I use these terms interchangeably here.

In this article, you'll learn the basics of BPF syntax and also see how it works under the hood.

The Testbed

To get a better idea of what BPF really is, let's go through a set of examples, generating series of packets each time and filtering them as needed. Sometimes these packets will come from a real network application, but other times you will craft them manually. It's not the best

idea to allow these forged packets out from the local host, especially if you are on the office network. So the first step will be to create a virtual Ethernet interface.

Linux already has a concept of “dummy” network interfaces, and the kernel module named `dummy` implements them. Load it, and assign the `dummy0` interface a unique IP address:

```
# modprobe dummy
# ip link set up dev dummy0
# ip addr add 192.168.2.1/24 dev dummy0
```

Next, you’ll need something to craft the packets and capture them subject to BPF filters. An obvious choice here is Scapy (<http://www.secdev.org/projects/scapy>), a Python toolkit for packet manipulation. Install it with your package manager or from the sources. Raw packet generation and live traffic capture are considered privileged operations in Linux, so you’ll need to run Scapy as root (for example, with `sudo`).

Scapy provides an interactive shell (which is naturally Python-based). You create different protocol layers as instances of the classes like TCP, IP or Ether. A complete list is available with the `ls()` command, and you can add your own protocol if

you really want to. The attributes of these classes correspond to protocol fields (addresses, ports, flags and so on). You can use raw numbers (say, 20) or symbolic names (`ftp_data`) for attribute values. To assemble the packet, use the `/` Python operator:

```
>>> Ether(src='08:60:6e:da:31:ae', dst='42:7f:79:88:de:3d') /
↳ IP(src='192.168.1.5', dst='192.168.2.1')
<Ether  dst=42:7f:79:88:de:3d src=08:60:6e:da:31:ae type=IPv4
↳ |<IP  src=192.168.1.5 dst=192.168.2.1 |>>
```

Protocol fields usually have sensible default values (you can check them with `ls(IP)` or similar), so you need to specify only those you want to override.

To disassemble the packet and get a specific protocol layer, use the `[]` operator:

```
>>> _[IP]
<IP  src=192.168.1.5 dst=192.168.2.1 |>
```

A special `_` variable contains the last expression’s value. Scapy makes it easy to generate a series of packets that follow a specific pattern:

```
>>> packets = Ether(src='08:60:6e:da:31:ae',
↳ dst='42:7f:79:88:de:3d') / IP(src='192.168.1.5',
↳ dst='192.168.2.1-3') / UDP(dport=[135, (137, 139)])
>>> len(list(packets))
```

12

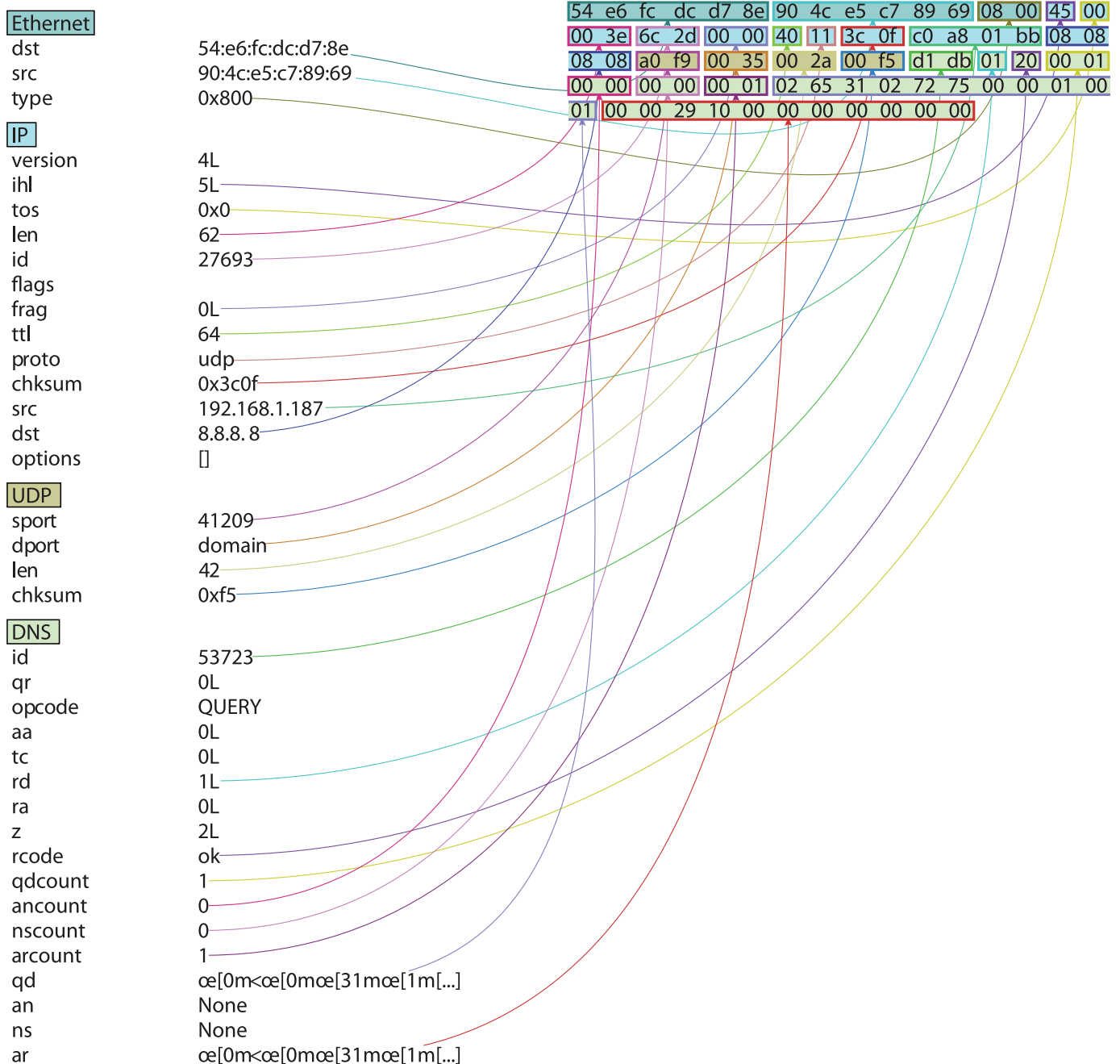


Figure 1. Scapy is a real Swiss army knife. It even can dump packets to PS or PDF.

Here, Scapy crafts 12 packets targeting UDP ports 135, 137, 138 and 139 (common Windows stuff) on three hosts. You can define address ranges with an asterisk (`dst='192.168.*.2'`) or

whole subnets with CIDR notation (`dst='192.168.2.0/24'`).

Several functions send packets over the wire, but here, let's deal with `send()` and `sendp()`. The main difference is that `sendp()`

Some Scapy Tips

You easily can convert between symbolic names and their numeric values using the `s2i` and `i2s` attributes defined on the protocol fields. Both are dictionaries, which you can use to look up the mapping:

```
>>> TCP.sport.s2i['www'], TCP.sport.i2s[21]
(80, 'ftp')
```

Note that I've used a class attribute here (not the instance one):

```
>>> TCP().sport
20
```

This is the default value for the TCP source port in Scapy:

```
>>> ls(TCP)
sport          : ShortEnumField      = (20)
...
```

Many network protocol fields also accept bit flags. Scapy allows single letter mnemonics for them:

```
>>> TCP().flags= 'S' # TCP SYN packet
```

Available mnemonics are in the field's "names" attribute:

```
>>> TCP().flags.names
'FSRPAUEC'
```

works on Layer 2, so the packet you pass it must have an Ethernet header. `send()` is for Layer 3, and it looks up the host's routing table to decide what to do with the packet you gave it:

```
>>> sendp(packets, iface='dummy0')
.....
Sent 12 packets.
```

This way, you push crafted packets into the `dummy0` interface.

To capture packets, use the `sniff()` function. It has many different options, but on these pages, you'll mainly call it like this:

```
>>> sniff(iface='dummy0', filter='udp')
^C<Sniffed: TCP:0 UDP:12 ICMP:0 Other:0>

>>> _.summary()

Ether / IP / UDP 192.168.1.5:domain > 192.168.2.1:epmap
Ether / IP / UDP 192.168.1.5:domain > 192.168.2.1:netbios_ns
... 10 lines skipped ...
```

If you omit the `iface=`, Scapy will listen on all network interfaces. You also can add the `count=` argument to capture only as many packets as specified; otherwise, you should stop the capture manually with `Ctrl-C`. Without the `filter=`, `sniff()` captures all packets. Internally, Scapy uses `libpcap` to compile the filter (either directly or via the

tcpdump -ddd command), so the syntax is just what you want.

This was a quick tour of Scapy; however, this tool can do much more than you've seen so far. Consult the official documentation (<http://www.secdev.org/projects/scapy/doc>) for more information. Or, have a look at *Security Power Tools* published by O'Reilly (2007), which has a complete chapter (number 6) on Scapy written by its author, Philippe Biondi.

Let's Filter

Now with the tools to experiment with in place, it's time to learn some actual high-level BPF. The most authoritative (and complete) reference documentation is the tcpdump(1) man page. Let's do a quick summary: filter expressions contain one or more primitives combined with the "and", "or" and "not" keywords (equivalently written as &&, || and !). All basic arithmetic and bit operations are supported as well, and the explicit precedence can be set using parentheses. If you omit parentheses, && and || are of the same precedence, and ! is applied first. Arithmetic and bit operations follow common rules.

Each primitive consists of the ID (either name or number) preceded with an optional protocol (ether, ip,

tcp or udp, to name a few), direction (src, dst, src or dst, and src and dst; it is always a single token, so or and and aren't operators) and type (host, net, port or portrange). If some of these are missing, all protocols match; host is assumed for the type, and src or dst for the direction (this means either direction is okay).

The following are valid primitives: udp (no ID here), port 80 (TCP or UDP port 80), ip host pluto (all IP packets for the host "pluto"; the name must be resolvable), dst tcp portrange 0-1023 (packets targeting all privileged TCP ports).

That's enough theory—now let's play. Arguably the most common case is to filter packets by their source or destination IP addresses. Let's use one Scapy instance to generate empty IP datagrams for random (and even nonexistent) hosts:

```
>>> while True: sendp(Ether(src=RandMAC(), dst=RandMAC()) /  
    ➡ IP(src=RandIP(), dst=RandIP()) / ICMP(), iface='dummy0')
```

There are many Rand*() functions in Scapy; two of them are used here.

A second Scapy instance will capture the datagrams that happen to be for host 192.168.1.1 or 192.168.1.2 (note that for the second operand in the expression, the host

keyword is implicit):

```
>>> sniff(iface='dummy0', filter='host 192.168.1.1 or
↳192.168.1.2', count=1)
```

Unless you are very lucky, `sniff()` will hang until you press Ctrl-C. That's because the probability for a given two addresses to occur in a randomly generated packet series is around 0.0000001%. Let's increase your chances and look for a whole A-class subnet:

```
>>> sniff(iface='dummy0', filter='net 1.0.0.0/8', count=1)
```

Equivalently, you can rewrite the filter as `net 1.0.0.0 mask 255.0.0.0`. This time, `sniff()` should catch a packet much sooner:

```
>>> _.summary()
Ether / 185.0.19.206 > 1.205.135.116 hopopt
```

It's equally easy to filter traffic on ports. For instance, this is a very simple filter for MySQL client-server sessions:

```
>>> sniff(iface='lo', filter='tcp port 3306')
```

Try to connect to the MySQL server with the `mysql -h 127.0.0.1 -P 3306` command, and you'll see some packets captured. It doesn't matter if the server is actually running—if

it doesn't, you'll catch a pair of TCP SYN and RST-ACK indicating the port is unavailable:

```
>>> _.summary()
Ether / IP / TCP 127.0.0.1:57485 > 127.0.0.1:mysql S
Ether / IP / TCP 127.0.0.1:mysql > 127.0.0.1:57485 RA
```

Detecting services by the port numbers they use isn't really accurate. Let's do a deeper packet analysis. Imagine you need to collect data pertaining to NTP activity on the local network. NTP messages are sent to (or from, or both) UDP port 123. Moreover, they are 48 bytes in size and have the status of a local clock, which can't be greater than 4, encoded in bits 2–7 of the first octet (see Appendix B in RFC 958 at <http://tools.ietf.org/html/rfc958>). Given all of that, the filter you construct will consist of three primitives: `udp port 123` (for the first two conditions), and the other two checking the UDP payload length and the clock status value.

To implement them, let's peek inside a protocol payload. BPF uses the `[offset:size]` operator for this purpose. Offset is measured from byte 0, so for instance, `tcp[0]` gives the first byte of the TCP header (not a payload). The UDP datagram Length field is at offset 4 and is 16 bits wide. Thus,

to check the packet length, you can use the following primitive: `udp[4:2] == 48+8` (UDP header length 8 is bytes, and they are included in the Length field as well). The clock status check is a bit more convoluted, but the combined filter looks like this:

```
>>> ntp = sniff(filter='proto udp and port 123
↳and udp[4:2] == 48+8 and ((udp[8] & 0x38) >> 3) <= 4')
```

Note the operators and hex numbers used—BPF is much like C in this sense. Leave this alone for some minutes, and on a modern Linux distribution you'll sooner or later spot the NTP requests the system sends (and the replies it receives):

```
>> ntp.summary()
Ether / IP / UDP / NTP v4, client
Ether / IP / UDP / NTP v4, server
...
>> ntp[0][NTP]
<NTP leap=nowarning version=4L mode=client stratum=2L
↳poll=10L precision=233L delay=0.0422210693359
↳dispersion=0.0782623291016 id=194.190.168.1 ref=Tue,
↳15 Apr 2014 10:00:26 +0000 orig=Tue, 15 Apr 2014
↳10:15:20 +0000 recv=Tue, 15 Apr 2014 10:15:20
↳+0000 sent=Tue, 15 Apr 2014 10:33:04 +0000 |>
```

In fact, it works so well, you even can drop the `udp proto 123` part

without introducing too much error (my quick tests show about a 1% rate).

Another way to get the packet length is to use the `len` operator. However, it accounts for all protocols used down to Layer 2. So, assuming UDP datagrams are encapsulated in IPv4 with no options and untagged Ethernet frames, you can rewrite `udp[4:2] == 48+8` as `len == 14+20+8+48`. By the way, you also can select packets longer (or shorter) than a specific threshold with the `greater` and `less` keywords.

Now you've seen that Scapy can decode (and encode) some application-level protocols like NTP (or DNS). But it can dig even deeper, so for the next example, let's use it to forge VLAN-tagged traffic:

```
>>> sendp(Ether(src=RandMAC(), dst=RandMAC()) /
↳Dot1Q(vlan=[(1,5)]) / 'Nothing to see here', iface='dummy0')
```

This generates five 802.1Q Ethernet frames with arbitrary MAC addresses and VLAN tags in range 1–5 (note the syntax). Let's not worry about the payload now, so instead of IPv4 (or any other network-level) packet, let's just put raw bytes that form a 'Nothing to see here' string inside the frame.


```

val@vsnitsyn:~$ sudo scapy
INFO: Can't import python gnuplot wrapper. Won't be able to plot.
WARNING: No route found for IPv6 destination :: (no default route?)
Welcome to Scapy (2.2.0)
>>> Ether(src='08:00:00:00:00:00', dst='42:7f:79:88:de:3d')/IP(src='192.168.1.5', dst='192.168.2.1')
<Ether  dst=42:7f:79:88:de:3d  src=08:00:00:00:00:00  type=IPv4  |<IP  src=192.168.1.5  dst=192.168.2.1  |>>
>>> [IP]
<IP  src=192.168.1.5  dst=192.168.2.1  |>
>>> packets = Ether(src='08:00:00:00:00:00', dst='42:7f:79:88:de:3d')/IP(src='192.168.1.5', dst='192.168.2.1-3')/UDP(dport=[135,(137,139)])
>>> Send(packets, iface='dummy0')
.....
Sent 12 packets.
>>> sniff(iface='eth0', filter='udp port 53')
^C<Sniffed: TCP:0 UDP:2 ICMP:0 Other:0>
>>> results
>>> results.summary()
Ether / IP / UDP / DNS Qry "www.linuxjournal.com."
Ether / IP / UDP / DNS Ans "76.74.252.198"
>>> results[0][DNS]
<DNS  id=1637  qname=www.linuxjournal.com.  qtype=A  qclass=IN  |>
An=None  ns=None  ar=None  rr=None  rrtype=None  rrsig=None  rrlen=None  rrfixed=None  rrrfc253=None
>>> ls(DNS)
id      : ShortField      = (0)
qr      : BitField        = (0)
opcode  : BitEnumField    = (0)
aa      : BitField        = (0)
tc      : BitField        = (0)
rd      : BitField        = (0)
ra      : BitField        = (0)
z       : BitField        = (0)
rcode   : BitEnumField    = (0)
qdcount : DNSRRCountField = (None)
ancount : DNSRRCountField = (None)
nscount : DNSRRCountField = (None)
arcount : DNSRRCountField = (None)
qd       : DNSRRField      = (None)
an       : DNSRRField      = (None)
ns       : DNSRRField      = (None)
ar       : DNSRRField      = (None)
>>> DNS.opcode.s2l.keys()
['STATUS', 'QUERY', 'IQUERY']
>>> sri(results[0])
Begin emission:
WARNING: Mac address to reach destination not found. Using broadcast.
Finished to send 1 packets.
.....^C
Received 51 packets, got 0 answers, remaining 1 packets
>>> quit()
val@vsnitsyn:~$

```

Figure 2. Capture, craft, dissect and do other funky things with network packets in Scapy.

When the Engine Matters

In Linux, Scapy defaults to PF_PACKET sockets internally. And, to my experience, sometimes it grabs more than the filter permits. If this happens to you too, force Scapy to use libpcap as an engine. First, install the pypcap library (it is usually named python-pypcap or something similar in your package manager). Then, edit `scapy/config.py` (depending on how you installed Scapy, it may be under `/usr/lib/python2.7` or somewhere else), and set `Conf.use_pcap` to `True`.

This is how you can filter traffic with specific VLAN tags in BPF:

```

>>> sniff(iface='dummy0', filter='vlan 1', count=1)
<Sniffed: TCP:0 UDP:0 ICMP:0 Other:1>
>>> _
<Ether  dst=29:a0:ea:e9:df:ce  src=c3:33:a3:9c:63:9c
  type=n_802_1Q  |<Dot1Q  prio=0L  id=0L  vlan=1L
  type=0x0  |<Padding  load='Nothing to see here'  |>>>

```

If you just want to filter out untagged traffic, use `filter='vlan'`.

Under the Hood

As you already know, BPF filters are actually expressed in a low-level assembler-like language. It targets

a register-based virtual machine that has an accumulator register and an index register, along with some memory store. This machine has access to the packet buffer and supports several dozens of instructions that store and load values to the registers or memory, perform arithmetic or logical operations and do flow control. I won't discuss them in detail, but if you want a taste of what they look like, here is the equivalent of a libpcap-compiled simple filter, ip:

```
ldh [12]
jeq #0x800, Keep, Drop
Keep: ret #0xffff
Drop: ret #0x0000
```

Opcode mnemonics come from the original BPF USENIX paper (Steven McCanne and Van Jacobson's "The BSD Packet Filter: A New Architecture for User-level Packet Capture" available at <https://www.usenix.org/legacy/publications/library/proceedings/sd93/mccanne.pdf>). This program loads 16-bit (short) integer ("h" stands for "half-word") at the fixed offset in the packet buffer (12, the Ethernet Type field) and compares it to 0x800 (the IPv4 protocol number). The filter returns the number of bytes in the packet to allow. Thus, zero means the

packet should be dropped, and 0xffff (the maximum possible length) means to accept it entirely.

Linux's own implementation of BPF design is known as Linux Socket Filtering (LSF). It differs from the original BPF slightly (mainly in areas I haven't mentioned in this article), but largely what I've said here about BPF applies to LSF.

Given that the virtual machine is register-based, it comes as no surprise that its bytecode easily can be compiled to real machine instructions. It is exactly what a BPF JIT compiler (introduced in Linux 3.0) does. On my x86_64 system, the code above compiles into:

```
0:  push    %rbp
1:  mov     %rsp,%rbp
4:  sub     $0x60,%rsp
8:  mov     %rbx,-0x8(%rbp)
c:  mov     0x68(%rdi),%r9d
10:  sub     0x6c(%rdi),%r9d
14:  mov     0xd8(%rdi),%r8
1b:  mov     $0xc,%esi
20:  callq   0xfffffffffe104b2b3
25:  cmp     $0x800,%eax
2a:  jne     0x0000000000000033
2c:  mov     $0xffff,%eax
31:  jmp     0x0000000000000035
33:  xor     %eax,%eax
35:  leaveq
36:  retq
```

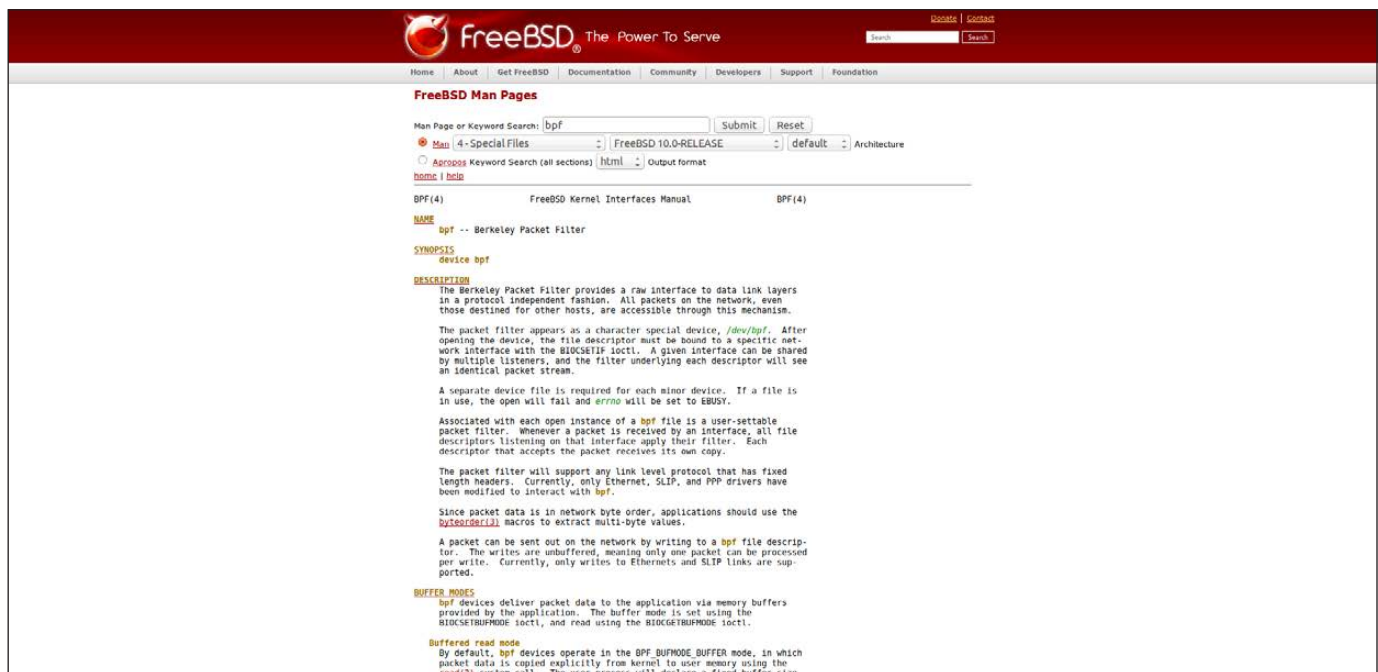


Figure 3. BPF and LSF are not the same; however, you still can use BSD-originated man pages.

This is a small function. By convention, the return value is stored in %eax register. Everything up to offset 0x1b is the utility stuff, but you easily can spot loading 0xc (12) to %esi (x86 source index register) and compare the result to 0x800. A few dozens of machine-level instructions are needed to check the filter against a packet, and since BPF was designed to execute filters on the original packet and not a copy of it, this should really be a fast process.

To control BPF JIT in the Linux kernel, you write to /proc/sys/net/core/bpf_jit_enable (or use the corresponding sysctl—see Jonathan Corbet’s “A JIT for packet filters”

at <https://lwn.net/Articles/437981>). A value of 1 means to enable JIT, and a value of 2 means to enable tracing. When tracing is enabled, JIT code for the filter is printed to the kernel log buffer (that’s what dmesg reads). With this in place, you can use the bpf_disasm tool (it comes with the Linux kernel sources) to print a disassembly.

With the neat high-level language libpcap provides, it is unlikely you will need to program in BPF “assembler” directly. But if you ever want some BPF feature that libpcap doesn’t provide (maybe filtering by Netfilter mark value or something else Linux-specific), take a look at netsniff-ng (<http://netsniff-ng.org>). This toolkit

contains a fully functional BPF compiler, `bpfc`.

Finally, what if you want to integrate BPF filtering capabilities into your own code? If it isn't possible (or feasible) to link against libpcap (or use one of its bindings), you can call the native kernel interface directly. BPF filter programs are represented as `struct sock_fprog` that has a pointer to the array of opcodes (`struct sock_filter *`) and a program length field. A `sock_filter` structure is what `tcpdump -dd` (or `bpfc`) prints for you:

```
# tcpdump -i dummy0 -dd ip
{ 0x28, 0, 0, 0x0000000c },
{ 0x15, 0, 1, 0x00000800 },
{ 0x6, 0, 0, 0x0000ffff },
{ 0x6, 0, 0, 0x00000000 },
```

Sometimes (for example, in a script) you even can execute this command in the runtime (it is exactly what Scapy does when `conf.use_pcap` is `False`).

Filters are attached to (and detached from) the socket with the `setsockopt(2)` system call:

```
setsockopt(sockfd, SOL_SOCKET, SO_ATTACH_FILTER,  
    &bpf, sizeof(bpf));
```

Here, `sockfd` is the socket descriptor and `bpf` is the struct `sock_fprog`.

BPF for Non-Sockets

So far, I've spoken of BPF (and LSF) as a socket-level facility. To conclude this article, let's look at it from a different angle. Starting with Linux 3.9, there is an `xt_bpf` module that allows you to use BPF in `iptables` rules:

```
iptables -A INPUT -m bpf --bytecode "$(nfbpf_compile
➡RAW 'tcp dst port telnet')" -j DROP
```

The `nfbpf_compile` utility comes with `iptables`, provided those were built with the `--enable-bpf-compiler` flag to the `configure` script.

Some tests show that `xt_bpf` performs even faster than the `u32` `xtables` module, so it's worth considering when you optimize your firewall rules.

Berkeley Packet Filters and their OS-specific implementations are no substitute for conventional firewalling code (like Netfilter in Linux). However, they may become indispensable when a fast application or system-level traffic filtering is the requirement. ■

Valentine Sinitsyn spent years developing easy-to-use Linux-based network solutions for a local SMB market. In his spare time, he teaches Physics.

|||||

Send comments or feedback via
<http://www.linuxjournal.com/contact>
or to ljeditor@linuxjournal.com.

WEBCASTS



Learn the 5 Critical Success Factors to Accelerate IT Service Delivery in a Cloud-Enabled Data Center

Today's organizations face an unparalleled rate of change. Cloud-enabled data centers are increasingly seen as a way to accelerate IT service delivery and increase utilization of resources while reducing operating expenses. Building a cloud starts with virtualizing your IT environment, but an end-to-end cloud orchestration solution is key to optimizing the cloud to drive real productivity gains.

> <http://lnxjr.nl/IBM5factors>



Modernizing SAP Environments with Minimum Risk—a Path to Big Data

Sponsor: SAP | **Topic:** Big Data

Is the data explosion in today's world a liability or a competitive advantage for your business? Exploiting massive amounts of data to make sound business decisions is a business imperative for success and a high priority for many firms. With rapid advances in x86 processing power and storage, enterprise application and database workloads are increasingly being moved from UNIX to Linux as part of IT modernization efforts. Modernizing application environments has numerous TCO and ROI benefits but the transformation needs to be managed carefully and performed with minimal downtime. Join this webinar to hear from top IDC analyst, Richard Villars, about the path you can start taking now to enable your organization to get the benefits of turning data into actionable insights with exciting x86 technology.

> <http://lnxjr.nl/modsap>

WHITE PAPERS



White Paper: JBoss Enterprise Application Platform for OpenShift Enterprise

Sponsor: DLT Solutions

Red Hat's® JBoss Enterprise Application Platform for OpenShift Enterprise offering provides IT organizations with a simple and straightforward way to deploy and manage Java applications. This optional OpenShift Enterprise component further extends the developer and manageability benefits inherent in JBoss Enterprise Application Platform for on-premise cloud environments.

Unlike other multi-product offerings, this is not a bundling of two separate products. JBoss Enterprise Middleware has been hosted on the OpenShift public offering for more than 18 months. And many capabilities and features of JBoss Enterprise Application Platform 6 and JBoss Developer Studio 5 (which is also included in this offering) are based upon that experience.

This real-world understanding of how application servers operate and function in cloud environments is now available in this single on-premise offering, JBoss Enterprise Application Platform for OpenShift Enterprise, for enterprises looking for cloud benefits within their own datacenters.

> <http://lnxjr.nl/jbossapp>

WHITE PAPERS



Linux Management with Red Hat Satellite: Measuring Business Impact and ROI

Sponsor: **Red Hat** | Topic: **Linux Management**

Linux has become a key foundation for supporting today's rapidly growing IT environments. Linux is being used to deploy business applications and databases, trading on its reputation as a low-cost operating environment. For many IT organizations, Linux is a mainstay for deploying Web servers and has evolved from handling basic file, print, and utility workloads to running mission-critical applications and databases, physically, virtually, and in the cloud. As Linux grows in importance in terms of value to the business, managing Linux environments to high standards of service quality — availability, security, and performance — becomes an essential requirement for business success.

> <http://lnxjr.nl/RHS-ROI>



Standardized Operating Environments for IT Efficiency

Sponsor: **Red Hat**

The Red Hat® Standard Operating Environment SOE helps you define, deploy, and maintain Red Hat Enterprise Linux® and third-party applications as an SOE. The SOE is fully aligned with your requirements as an effective and managed process, and fully integrated with your IT environment and processes.

Benefits of an SOE:

SOE is a specification for a tested, standard selection of computer hardware, software, and their configuration for use on computers within an organization. The modular nature of the Red Hat SOE lets you select the most appropriate solutions to address your business' IT needs.

SOE leads to:

- Dramatically reduced deployment time.
- Software deployed and configured in a standardized manner.
- Simplified maintenance due to standardization.
- Increased stability and reduced support and management costs.
- There are many benefits to having an SOE within larger environments, such as:
 - Less total cost of ownership (TCO) for the IT environment.
 - More effective support.
 - Faster deployment times.
 - Standardization.

> <http://lnxjr.nl/RH-SOE>

Security Hardening with Ansible

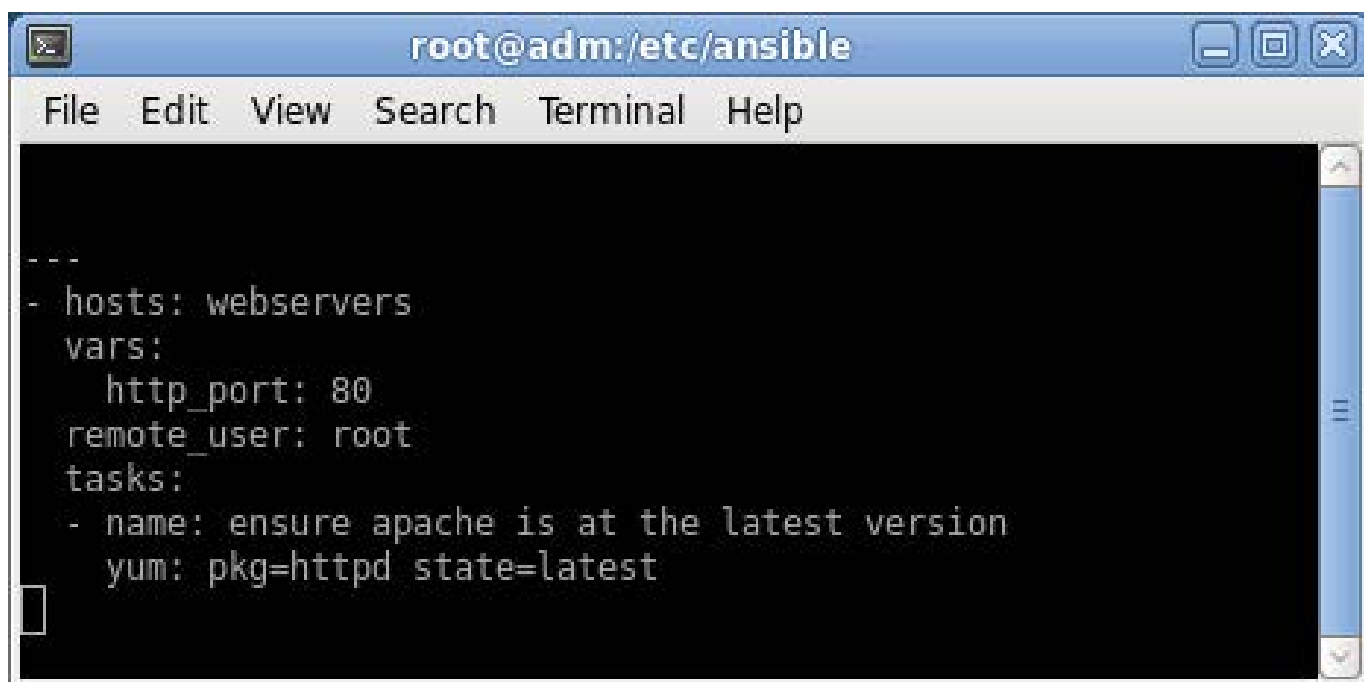
Learn how you can harden your RHEL6 systems quickly and efficiently.

MARK DOTSON

Ansible is an open-source automation tool developed and released by Michael DeHaan and others in 2012. DeHaan calls it a “general-purpose automation pipeline” (see Resources for a link to the article “Ansible’s Architecture: Beyond Configuration Management”). Not only can it be used for automated configuration management, but it also excels at orchestration, provisioning of systems, zero-time rolling updates and application deployment. Ansible can be used to keep all your systems configured exactly the way you want them, and if you have many identical systems, Ansible will ensure they stay identical. For Linux system administrators, Ansible is an indispensable tool in implementing and maintaining a strong security posture.

Ansible can be used to deploy and configure multiple Linux servers (Red Hat, Debian, CentOS, OS X, any of the BSDs and others) using secure shell (SSH) instead of the more common client-server methodologies used by other configuration management packages, such as Puppet and Chef (Chef does have a solo version that does not require a server, per se). Utilizing SSH is a more secure method because the traffic is encrypted. The secure shell transport layer protocol is used for communications between the Ansible server and the target hosts. Authentication is accomplished using Kerberos, public-key authentication or passwords.

When I began working in system administration some years ago, a senior colleague gave me a simple formula for success. He said, “Just

A screenshot of a terminal window titled 'root@adm:/etc/ansible'. The window has a menu bar with 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The terminal content shows an Ansible playbook in YAML format. It starts with '---', followed by 'hosts: webservers', 'vars:' with 'http_port: 80' and 'remote_user: root', and 'tasks:' with a single task 'name: ensure apache is at the latest version' and 'yum: pkg=httpd state=latest'.

```
---  
- hosts: webservers  
  vars:  
    http_port: 80  
    remote_user: root  
  tasks:  
    - name: ensure apache is at the latest version  
      yum: pkg=httpd state=latest
```

Figure 1. Example Playbook That Will Upgrade Apache to the Latest Version

remember, automate, automate, automate.” If this is true, and I believe it is, then Ansible can be a crucial tool in making any administrator’s career successful. If you do not have a few really good automation tools, every task must be accomplished manually. That wastes a lot of time, and time is precious. Ansible makes it possible to manage many servers almost effortlessly.

Ansible uses a very simple method called playbooks to orchestrate configurations. A playbook is a set of instructions written in YAML that tells the Ansible server what “plays” to carry out on the target hosts. YAML is a very simple, human-readable

markup language that gives the user fine granularity when setting up configuration schemes. It is installed, along with Ansible, as a dependency. Ansible uses YAML because it is much easier to write than common data formats, like JSON and XML. The learning curve for YAML is very low, hence proficiency can be gained very quickly. For example, the simple playbook shown in Figure 1 keeps the Apache RPM on targeted Web servers up to date and current.

From the Ansible management server, you can create a cron job to push the playbook to the target hosts on a regular basis, thus ensuring you always will have the

latest-and-greatest version of the Apache Web server.

Using YAML, you can instruct Ansible to target a specific group of servers, the remote user you want to run as, tasks to assign and many other details. You can name each task, which makes for easier reading of the playbook. You can set variables, and use loops and conditional statements. If you have updated a configuration file that requires restarting a service, Ansible uses tasks called handlers to notify the system that a service restart is necessary. Handlers also can be used for other things, but this is the most common.

The ability to reuse certain tasks from previously written playbooks is another great feature. Ansible uses a mechanism called roles to accomplish this. Roles are organizational units that are used to implement a specific configuration on a group of hosts. A role can include a set of variable values, handlers and tasks that can be assigned to a host group, or hosts corresponding to specific patterns. For instance, you could create a role for installing and configuring MySQL on a group of targeted servers. Roles make this a very simple task.

Besides intelligent automation, you also can use Ansible for ad

hoc commands to contact all your target hosts simultaneously. Ad hoc commands can be performed on the command line. It is a very quick method to use when you want to see a specific type of output from all your target machines, or just a subset of them. For example, if you want to see the uptime for all the hosts in a group called dbservers, you would type, as user root:

```
# ansible dbservers -a /usr/bin/uptime
```

The output will look like Figure 2.

If you want to specify a particular user, use the command in this way:

```
# ansible dbservers -a /usr/bin/uptime -u username
```

If you are running the command as a particular user, but want to act as root, you can run it through sudo and have Ansible ask for the root password:

```
# ansible dbservers -a /usr/bin/uptime -u username
└─[--sudo [ask-sudo-pass]]
```

You also can switch to a different user by using the -U option:

```
# ansible dbservers -a /usr/bin/uptime -u username
└─[-U otheruser --sudo]
# [ask-sudo-pass]
```

```

root@adm:/
File Edit View Search Terminal Help
[root@adm /]# ansible hosts -a uptime
192.168.0.101 | success | rc=0 >>
 10:06:43 up 3 days, 21:21,  0 users,  load average: 0.00, 0.00, 0.00

192.168.0.102 | success | rc=0 >>
 10:06:43 up 3 days, 21:21,  0 users,  load average: 0.04, 0.01, 0.00

192.168.0.105 | success | rc=0 >>
 10:06:43 up 3 days, 21:21,  0 users,  load average: 0.00, 0.00, 0.00

192.168.0.104 | success | rc=0 >>
 10:06:43 up 3 days, 21:21,  0 users,  load average: 0.00, 0.00, 0.00

192.168.0.103 | success | rc=0 >>
 10:06:43 up 3 days, 21:21,  0 users,  load average: 0.00, 0.00, 0.00

192.168.0.106 | success | rc=0 >>
 10:06:43 up 3 days, 21:21,  0 users,  load average: 0.00, 0.00, 0.00

192.168.0.107 | success | rc=0 >>
 10:06:43 up 3 days, 21:21,  0 users,  load average: 0.00, 0.00, 0.00

192.168.0.108 | success | rc=0 >>
 10:06:43 up 3 days, 21:23,  0 users,  load average: 0.00, 0.00, 0.00

[root@adm /]#

```

Figure 2. Example of ad hoc Command Showing Uptime Output for All Targets

Occasionally, you may want to run the command with 12 parallel forks, or processes:

```
# ansible dbservers -a /usr/bin/uptime -f 12
```

This will get the job done faster by using 12 simultaneous processes, instead of the default value of 5. If you would like to set a permanent default for the number of forks, you can set it in the Ansible configuration file, which

is located in `/etc/ansible/ansible.cfg`.

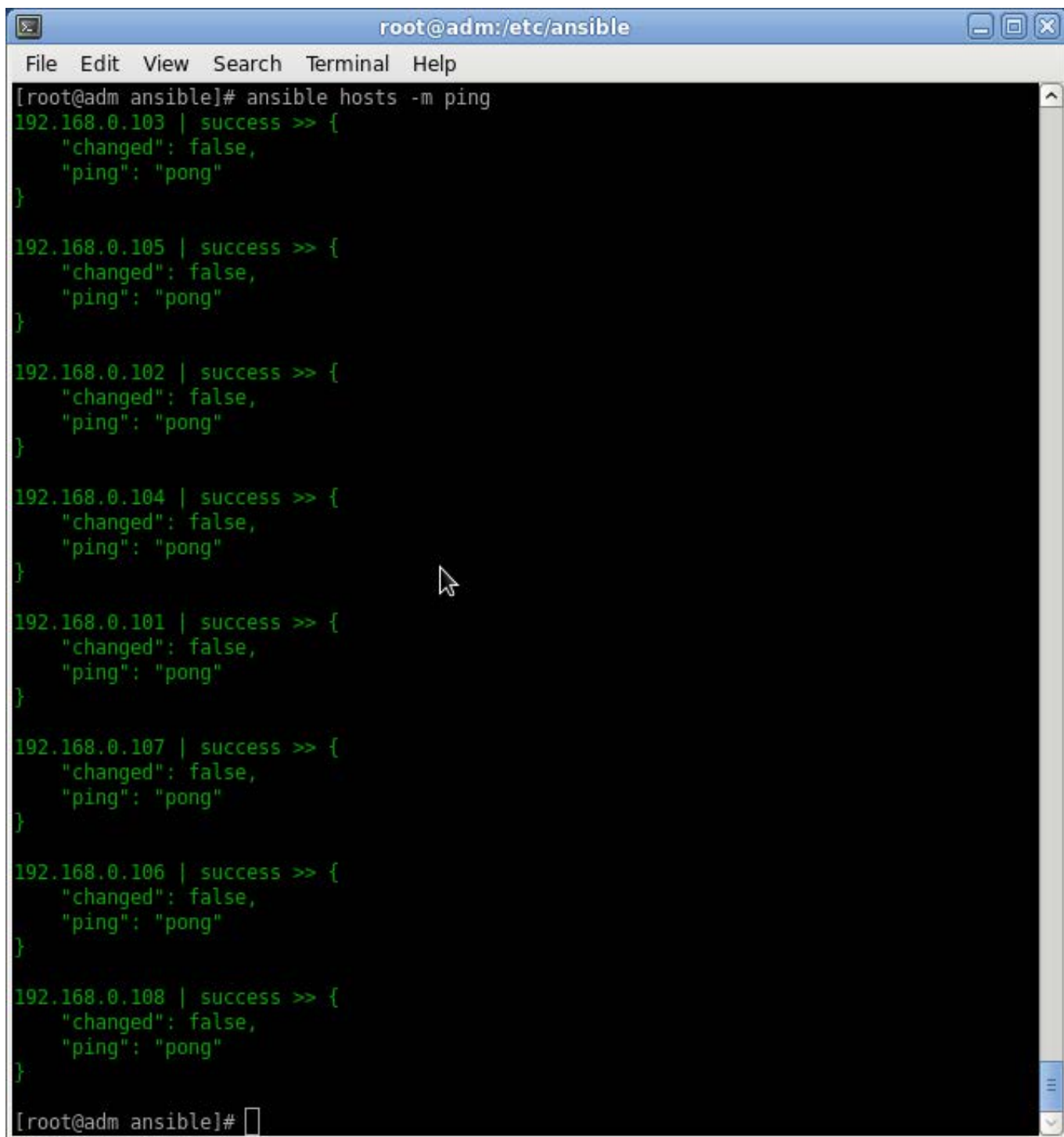
It also is possible to use Ansible modules in ad hoc mode by using the `-m` option. In this example, Ansible pings the target hosts using the `ping` module:

```
# ansible dbservers -m ping
```

As I write this, Michael DeHaan has announced that, in a few weeks, a new command-line tool will be

added to Ansible version 1.5 that will enable the encrypting of various data within the configuration. The new

tool will be called `ansible-vault`. It will be implemented by using the new `--ask-vault-pass` option.

A screenshot of a terminal window titled 'root@adm:/etc/ansible'. The window has a menu bar with 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The terminal content shows the command '[root@adm ansible]# ansible hosts -m ping' being executed. The output displays the results of pinging eight different IP addresses: 192.168.0.103, 192.168.0.105, 192.168.0.102, 192.168.0.104, 192.168.0.101, 192.168.0.107, 192.168.0.106, and 192.168.0.108. Each line shows 'success' and a JSON object with 'changed': false and 'ping': 'pong'. The terminal ends with a prompt '[root@adm ansible]#'.

```
root@adm:/etc/ansible
File Edit View Search Terminal Help
[root@adm ansible]# ansible hosts -m ping
192.168.0.103 | success >> {
  "changed": false,
  "ping": "pong"
}
192.168.0.105 | success >> {
  "changed": false,
  "ping": "pong"
}
192.168.0.102 | success >> {
  "changed": false,
  "ping": "pong"
}
192.168.0.104 | success >> {
  "changed": false,
  "ping": "pong"
}
192.168.0.101 | success >> {
  "changed": false,
  "ping": "pong"
}
192.168.0.107 | success >> {
  "changed": false,
  "ping": "pong"
}
192.168.0.106 | success >> {
  "changed": false,
  "ping": "pong"
}
192.168.0.108 | success >> {
  "changed": false,
  "ping": "pong"
}
[root@adm ansible]#
```

Figure 3. In this example, Ansible pings the target hosts using the ping module.

According to DeHaan, anything you write in YAML for your configuration can be encrypted with `ansible-vault` by using a password.

Server security hardening is crucial to any IT enterprise. We must face the fact that we are protecting assets in what has become an informational war-zone. Almost daily, we hear of enterprise systems that have fallen prey to malevolent individuals. Ansible can help us, as administrators, protect our systems. I have developed a very simple way to use Ansible, along with an open-source project called `Aqueduct`, to harden RHEL6 Linux servers. These machines are secured according to the standards formulated by the Defense Information Systems Agency (DISA). DISA publishes Security Technical Implementation Guides (STIGs) for various operating systems that provide administrators with solid guidelines for securing systems.

In a typical client-server setup, the remote client `dæmon` communicates with a server `dæmon`. Usually, this communication is in the clear (not encrypted), although Puppet and Chef have their own proprietary mechanisms to encrypt traffic. The implementation of public-key authentication (PKI) in SSH has been well vetted for many years by security professionals and system

administrators. For my purposes, SSH is strongly preferred. Typically, there is a greater risk in using proprietary client-server `dæmons` than using SSH. They may be relatively new and could be compromised by malevolent individuals using buffer-overflow attack strategies or denial-of-service attacks. Any time we can reduce the total number of services running on a server, it will be more secure.

To install the current version of Ansible (1.4.3 at the time of this writing), you will need Python 2.4 or later and the Extra Packages for Enterprise Linux (EPEL) repository RPM. For the purposes of this article, I use Ansible along with another set of scripts from an open-source project called `Aqueduct`. This is not, however, a requirement for Ansible. You also will need to install Git, if you are not already using it. Git will be used to pull down the `Aqueduct` package.

Vincent Passaro, Senior Security Architect at Fotis Networks, pilots the `Aqueduct` project, which consists of the development of both bash scripts and Puppet manifests. These are written to deploy the hardening guidelines provided in the STIGs. Also included are CIS (Center for Internet Security) benchmarks and several others. On the `Aqueduct` home page, Passaro says, "Content is currently being developed

(by me) for the Red Hat Enterprise Linux 5 (RHEL 5) Draft STIG, CIS Benchmarks, NISPOM, PCI", but I have found RHEL6 bash scripts there as well. I combined these bash scripts to construct a very basic Ansible playbook to simplify security hardening of RHEL6 systems. I accomplished this by using the included Ansible module called script.

According to the Ansible documentation, "The script module takes the script name followed by a list of space-delimited arguments. The local script at path will be transferred to the remote node and then executed. The given script will be processed through the shell environment on the remote node. This module does not require Python on the remote system, much like the raw module."

Ansible modules are tiny bits of code used for specific purposes by the API to carry out tasks. The documentation states, "Ansible modules are reusable units of magic that can be used by the Ansible API, or by the `ansible` or `ansible-playbook` programs." I view them as being very much like functions or subroutines. Ansible ships with many modules ready for use. Administrators also can write modules to fit specific needs using any programming language. Many of the Ansible modules are idempotent, which means they will not make a

change to your system if a change does not need to be made. In other words, it is safe to run these modules repeatedly without worrying they will break things. For instance, running a playbook that sets permissions on a certain file will, by default, update the permissions on that file only if its permissions differ from those specified in the playbook.

For my needs, the script module works perfectly. Each Aqueduct bash script corresponds to a hardening recommendation given in the STIG. The scripts are named according to the numbered sections of the STIG document.

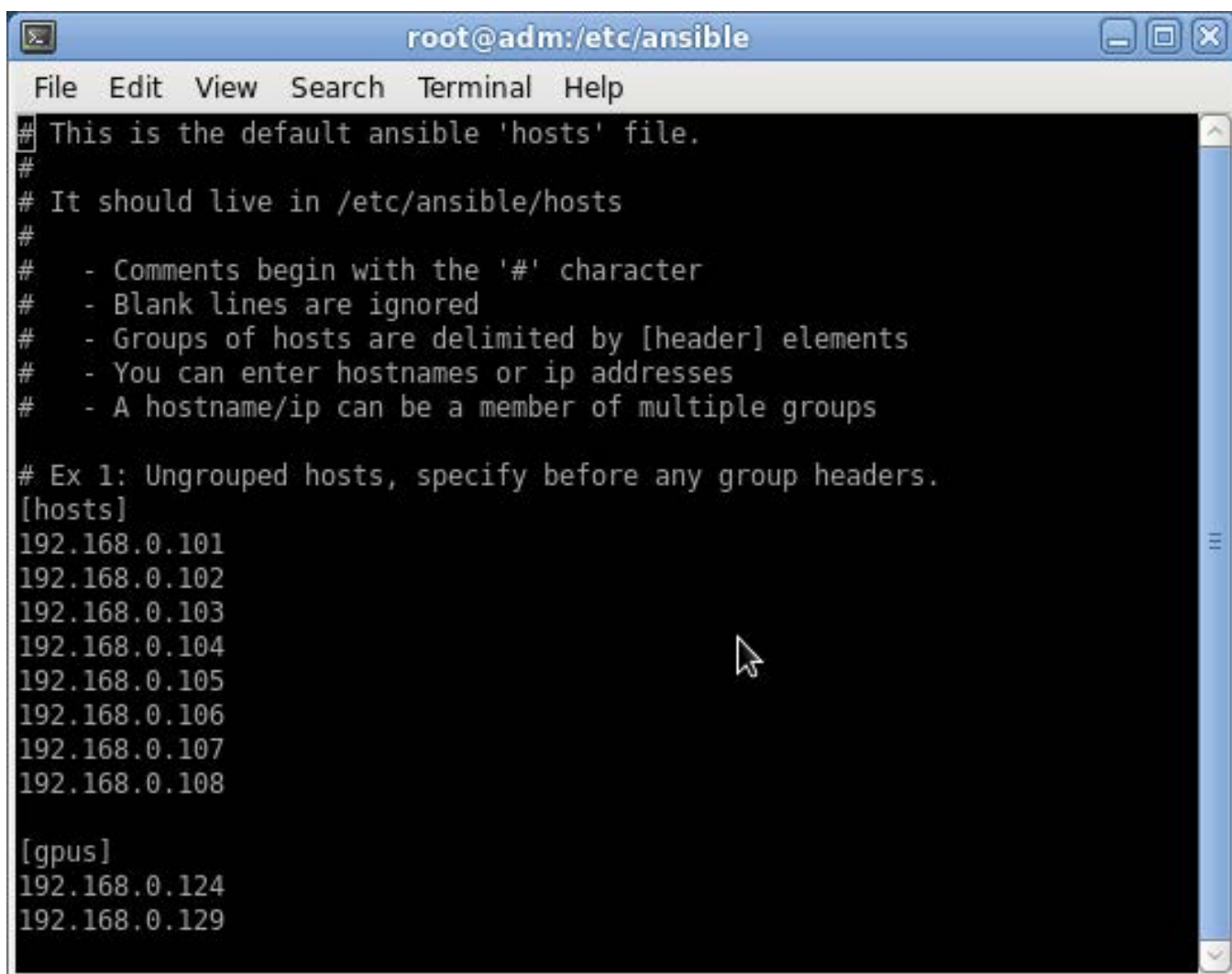
In my test environment, I have a small high-performance compute cluster consisting of one management node and ten compute nodes. For this test, the SSH server `dæmon` is configured for public-key authentication for the root user. To install Ansible on RHEL6, the EPEL repository must first be installed. Download the EPEL RPM from the EPEL site (see Resources).

Then, install it on your management node:

```
# rpm -ivh epel-release-6-8.noarch.rpm
```

Now, you are ready to install Ansible:

```
# yum install ansible
```



```
root@adm:/etc/ansible
File Edit View Search Terminal Help
# This is the default ansible 'hosts' file.
#
# It should live in /etc/ansible/hosts
#
# - Comments begin with the '#' character
# - Blank lines are ignored
# - Groups of hosts are delimited by [header] elements
# - You can enter hostnames or ip addresses
# - A hostname/ip can be a member of multiple groups
#
# Ex 1: Ungrouped hosts, specify before any group headers.
[hosts]
192.168.0.101
192.168.0.102
192.168.0.103
192.168.0.104
192.168.0.105
192.168.0.106
192.168.0.107
192.168.0.108
#
[gpus]
192.168.0.124
192.168.0.129
```

Figure 4. The /etc/hosts File for My Test Cluster

Ansible's main configuration file is located in `/etc/ansible/ansible.cfg`. Unless you want to add your own customizations, you can configure it with the default settings.

Now, create a directory in `/etc/ansible` called `prod`. This is where you will copy the Aqueduct STIG bash scripts. Also, create a directory in `/etc/ansible` called `plays`, where you will keep your Ansible playbooks. Create another

directory called `manual-check`. This will hold scripts with information that must be checked manually. Next, a hosts file must be created in `/etc/ansible`. It is simply called `hosts`. Figure 4 shows how I configured mine for the ten compute nodes.

Eight of the compute nodes are typical nodes, but two are equipped with GPGPUs, so there are two groups: "hosts" and "gpus". Provide

the IP address of each node (the host name also can be given if your DNS is set up properly). With this tiny bit of configuration, Ansible is now functional. To test it, use Ansible in ad hoc mode and execute the following command on your management node:

```
# ansible all -m ping
```

If this results in a “success” message from each host, all is well.

The Aqueduct scripts must be downloaded using Git. If you do not have this on your management node, then:

```
# yum install git
```

Git “is a distributed revision control and source code management (SCM) system with an emphasis on speed” (Wikipedia). The command-line for acquiring the Aqueduct package of scripts and manifests goes like this:

```
# git clone git://git.fedorahosted.org/git/aqueduct.git
```

This will create a directory under the current directory called aqueduct. The bash scripts for RHEL6 are located in aqueduct/compliance/bash/stig/rhel-6/prod. Now, copy all scripts therein to /etc/ansible/prod. There are some other aspects of the STIG that will

need to be checked by either running the scripts manually or reading the script and performing the required actions. These scripts are located in aqueduct/compliance/bash/stig/rhel-6/manual-check. Copy these scripts to /etc/ansible/manual-check.

Now that the scripts are in place, a playbook must be written to deploy them on all target hosts. Copy the playbook to /etc/ansible/plays. Make sure all scripts are executable. Figure 5 shows the contents of my simple playbook called aqueduct.yml.

On a few of the STIG scripts, a few edits were needed to get them to execute correctly. Admittedly, a more eloquent solution would be to replace the STIG scripts by translating them into customized Ansible modules. For now, however, I am taking the easier route by calling the STIG scripts as described from my custom Ansible playbook. The script module makes this possible. Next, simply execute the playbook on the management node with the command:

```
# ansible-playbook aqueduct.yml
```

This operation takes about five minutes to run on my ten nodes, with the understanding that the plays run in parallel on the target hosts. Ansible produces detailed output that shows the progress of each play and host. When


```

--
# This playbook applies DISA STIG hardening to all nodes via a series of bash scripts
# which are part of the Aqueduct project.
#
- hosts: all
  user: root
  tasks:
    - name: Aqueduct STIG scripts
      script: "{{ item }}"
      with_fileglob:
        - /etc/ansible/prod/RHEL*

```

Figure 5. My Simple Playbook to Execute STIG Scripts on All Targets

```

PLAY RECAP *****
192.168.0.101      : ok=2    changed=1    unreachable=0    failed=0
192.168.0.102      : ok=2    changed=1    unreachable=0    failed=0
192.168.0.103      : ok=2    changed=1    unreachable=0    failed=0
192.168.0.104      : ok=2    changed=1    unreachable=0    failed=0
192.168.0.105      : ok=2    changed=1    unreachable=0    failed=0
192.168.0.106      : ok=2    changed=1    unreachable=0    failed=0
192.168.0.107      : ok=2    changed=1    unreachable=0    failed=0
192.168.0.108      : ok=2    changed=1    unreachable=0    failed=0
192.168.0.124      : ok=2    changed=1    unreachable=0    failed=0
192.168.0.129      : ok=2    changed=1    unreachable=0    failed=0
[root@adm ansible]#

```

Figure 6. Output Showing a Successful STIG Playbook Execution

Ansible finishes running the plays, all of the target machines should be identically hardened, and a summary is displayed. In this case, everything ran successfully.

For system security hardening, the combination of Ansible and Aqueduct is a powerfully productive force in keeping systems safe from intruders.

If you've ever worked as a system administrator, you know how much time a tool like this can save. The more I learn about Ansible, the more useful it becomes. I am constantly thinking of new ways to implement it. As my system administration duties drift more toward using virtual technologies, I

plan on using Ansible to provision and manage my virtual configurations quickly. I am also looking for more avenues to explore in the way of managing high-performance computing systems, since this is my primary duty. Michael DeHaan has developed another tool called Cobbler, which is excellent for taking advantage of Red Hat's installation method, Kickstart, to build systems quickly. Together, Cobbler and Ansible create an impressive arsenal for system management.

As system administrators, we are living in exciting times. Creative developers are inventing an amazing array of tools that, not only make our jobs easier, but also more fun. I can only imagine what the future may hold. One thing is certain: we

will be responsible for more and more systems. This is due to the automation wizardry of technologies like Ansible that enable a single administrator to manage hundreds or even thousands of servers. These tools will only improve, as they have continued to do. As security continues to become more and more crucial, their importance will only increase. ■

Mark Dotson has been a system administrator for 15 years. He has worked in storage and high-performance computing. His hobbies include writing and reading philosophy. He is currently employed by Lockheed-Martin Corporation.

Send comments or feedback via
<http://www.linuxjournal.com/contact>
or to ljeditor@linuxjournal.com.

Resources

Ansible's Architecture: Beyond Configuration Management: <http://blog.ansibleworks.com/2013/11/29/ansible-architecture-beyond-configuration-management>

Michael DeHaan's Blog: <http://michaeldehaan.net>

Git Home: <http://git-scm.com>

Aqueduct Home: <http://www.vincentpassaro.com/open-source-projects/aqueduct-red-hat-enterprise-linux-security-development>

Ansible Documentation: <http://docs.ansible.com/index.html>

EPEL Repository Home: <https://fedoraproject.org/wiki/EPEL>

DISA RHEL6 STIG: http://iase.disa.mil/stigs/os/unix/red_hat.html

LINUXTM JOURNAL DevOps

With deep focus on
Collaborative Development,
Continuous Testing and
Release & Deployment,
we offer here the DEFINITIVE
DevOps for Dummies,
a **mobile Application**
Development Primer
plus advice and help from
expert sources like:

- Forrester
- Gartner
- IDC
- *Linux Journal*

Plus a host of other
eBooks, videos,
podcasts and more.

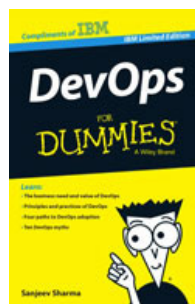
REGISTER NOW
and receive
unlimited access to
all site content and
downloads, plus
alerts when new
assets are made
available.

**Have projects in development
that need help?**

**Have a great development
operation in place that
can ALWAYS be better?**

Regardless of where you are in your
DevOps process, *Linux Journal* can help!

DevOps for Dummies

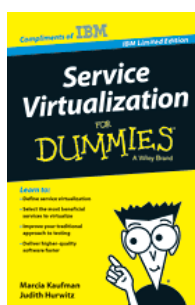


Free eBook

Today's fast-moving world makes DevOps essential for any business aspiring to be agile and lean in order to respond rapidly to changing customer and marketplace demands. This book helps you understand DevOps and how your organization can gain real business benefits from it.

You'll also discover how a holistic view of DevOps that encompasses the entire software delivery life cycle - from ideation and the conception of new business capabilities to implementation in production - can bring competitive advantage in a continuous delivery world.

Service Virtualization for Dummies Book



Free eBook

Discover service virtualization and how it fits into the big picture of software quality. In this book, Service Virtualization For Dummies, IBM Limited Edition, written by industry analysts Marcia Kaufman and Judith Hurwitz, learn how to deliver higher quality software by increasing the efficiency and effectiveness of your testing processes while reducing testing downtime and testing cost.

REGISTER NOW
<http://devops.linuxjournal.com>



DOC SEARLS

In the Matrix of Mobile, Linux Is Zion

In mobile we are losing the free world called the Web and the Net. How do we save it?

Already most of us spend more time on mobile devices than we do on desktops and laptops, put together. We also can do a lot more stuff, in a lot more places, on mobile devices than on computers. There were more than a million iOS apps on the shelves of Apple's store in October 2013 (<http://techcrunch.com/2013/06/10/apples-app-store-hits-50-billion-downloads-paid-out-10-billion-to-developers>), and I'm guessing there are at least that many Android apps on Google's shelves by now.

Meanwhile, app development on computers is slacking off—so is Web development, except as required to accessorize mobile apps. And on mobile devices, use of the Web is fading as well. According to Flurry Analytics

(<http://blog.flurry.com/bid/109749/Apps-Solidify-Leadership-Six-Years-into-the-Mobile-Revolution>), the Web's share of mobile use dropped from 20% in 2013 to 14% in 2014. In "The Decline of the Mobile Web" (<http://cdixon.org/2014/04/07/the-decline-of-the-mobile-web>), Chris Dixon writes:

This is a worrisome trend for the web. Mobile is the future. What wins mobile, wins the Internet. Right now, apps are winning and the web is losing.

Moreover, there are signs that it will only get worse. Ask any web company and they will tell you that they value app users

Underneath it all, the Internet is getting harder and harder to see, understand and appreciate.

more than web users. This is why you see so many popups and banners on mobile websites that try to get you to download apps. It is also why so many mobile websites are broken. Resources are going to app development over web development. As the mobile web UX further deteriorates, the momentum toward apps will only increase.

The likely end state is the web becomes a niche product used for things like 1) trying a service before you download the app, 2) consuming long tail content (e.g., link to a niche blog from Twitter or Facebook feed).

He sees an end state that “will probably be like cable TV—a few dominant channels/apps that sit on users’ home screens and everything else relegated to lower tiers or irrelevance”.

Those millions of apps are a forest of silos, growing on land that is privately owned or controlled by Apple, Google and Microsoft. Out on the streets, plains and hills of the civilized world, network connections

are also provided by mobile phone companies, whose own silos are walled by usage limits and by tariffs at national borders.

Underneath it all, the Internet is getting harder and harder to see, understand and appreciate. Already mobile operators in India are offering free or cheaper plans just for Facebook, Whatsapp and Twitter (<http://www.medianama.com/2013/01/223-airtel-vodafone-idea-data-internet-rates-increase>). To those operators, network neutrality means nothing. In fact, it never did, to any of the big operators. When he was Chief Scientist at BT, JP Rangaswami (<http://confusedofcalcutta.com>) said the core competence of phone companies is billing, not communications.

“Winning the Internet” should be absurd on its face, like “winning” sunlight or weather. But it isn’t in mobile, which already has turned into a giant Truman Show (http://en.wikipedia.org/wiki/The_Truman_Show). Inside that show, small app developers become suburbs of large ones—for example, by requiring logins through Facebook or Twitter, rather than using an identity

Many ordinary activities that are open in the physical world are now silo'd in the virtual one.

from the open world, such as an e-mail address. That's the case with Shazam (<http://www.shazam.com>), a handy mobile app you can use to identify the music you hear. But if you want to see the music you've "tagged" with Shazam on the company's Web site, the only way to log in is through Facebook. As for getting that data back out, good luck.

Even apps that do let you have your data back also make the process costly or difficult. Fitbit will give you XLS and CSV data, but only with a "premium" subscription of \$49.99/year. To get personal data gathered by Nike+ devices and apps, you'll need an external hack, such as `fuel_dump` (https://github.com/edrabbitt/fuel_dump). Integrating that data is also a steep challenge. Some app developers will partner with other companies or open their APIs to allow data to flow between apps, but integrating personal data gathered by mobile apps on a personal computer requires real wizardry. The Muggles won't bother.

There are exceptions though. My favorite is Moves (<http://moves-app.com>), a \$2.99 app from Finland that

produces an "activity diary" of your walking, running, cycling and riding around in the world. Its Web site kindly provides data exports in CSV, GEOSON, GEORSS, GPX, ICAL, JSONKML and KML_GE. Within those are Activities, Places and Storyline. The only loophole in its otherwise respectful privacy policy is one that lets your data go to a third party if that party buys the company (<http://www.moves-app.com/privacy>). Still, privacy laws are much stronger in Europe than in most other places, especially the US.

Many ordinary activities that are open in the physical world are now silo'd in the virtual one. Take reading books, for example. Buy a hardcover or paperback book, and it's yours. You can pull it off your shelf and read it without any company intermediating the process or telling you what you can and can't do. This is not the case with most electronic books, which can be read only on proprietary machines using proprietary software, both of which are mostly Amazon's. My wife just got a new Kindle Fire, as a gift from the airline she's flown for more than two million miles. It's a nice gift,

but using it feels like you're on an Amazon cruise ship with portholes looking out to the open world.

Even listening to radio on-line is an ordeal compared to what we had when stations could be found on radio dials. Although there are apps—TuneIn (<http://tunein.com>) and Wunderradio (<http://www.wunderradio.com>)—that try to replicate the dials of old (and it's not easy, given the vast numbers of stations streaming in the world), station owners feel the need to create an app for their own stations, or for fleets of them, to the exclusion of others. This is what Clear Channel (<http://www.clearchannel.com/Pages/Home.aspx>) does with iHeartRadio (<http://www.iheart.com>). It's a "dial" mostly of its own stations. To it, as to other large operators in the virtual world, the infrastructure that matters isn't the public stuff that's open and ownerless—such as the Net, the Web and e-mail—but what's private and controlled by companies with which deals can be done.

For example, right now at the top of the stream of tweets I see on Twitter is a "promoted" link—an ad—from Sprinklr. The tweet says, "FREE eBook: Without Infrastructure, You Can't Be Social". The link goes to a page about "The Rise of Social Experience Management", explained

Advertiser Index

Thank you as always for supporting our advertisers by buying their products!

ADVERTISER	URL	PAGE #
Drupalize.me	http://www.drupalize.me	15
EmperorLinux	http://www.emperorlinux.com	13
Linux Journal DevOps	http://devops.linuxjournal.com	7, 109
Silicon Mechanics	http://www.siliconmechanics.com	3
Texas Linux Fest	http://texaslinuxfest.org	2
USENIX Federated Conferences Week	https://www.usenix.org/conference/fcw14	41

ATTENTION ADVERTISERS

The *Linux Journal* brand's following has grown to a monthly readership nearly one million strong. Encompassing the magazine, Web site, newsletters and much more, *Linux Journal* offers the ideal content environment to help you reach your marketing objectives. For more information, please visit <http://www.linuxjournal.com/advertising>.

These completely private marketplaces are monopolies to mobile customers and monopsonies to app companies.

in a free eBook that requires giving up a bunch of personal data so you can become a qualified lead for whatever it is that Sprinklr sells. Note that the “infrastructure” is theirs. Not the Net’s. Not the Web’s. Not any space we all share. Just their space, which they own and charge rent to use.

Nowhere is mobile infrastructure more locked down and controlled than with the app stores. These completely private marketplaces are monopolies to mobile customers and monopsonies to app companies. As sole intermediaries, the stores get to charge what they want, which is a lot. Apple and Google both take 30% of what you pay for an app. So does Microsoft, but it drops its cut to 20% after sales of an app reach \$25k. These are monopoly rents. Ask your shoe or grocery store what kind of margins it gets—or hell, even Amazon.

I believe the only answer is to equip ordinary individuals (Muggles, not just wizards like us) with independent tools for integrating

and managing data and services—and better ways to exercise their autonomy, independence, freedom and agency in the world. I don't see any other way to break out of the Matrix (http://en.wikipedia.org/wiki/The_Matrix) that the mobile world is becoming.

We need to start with general-purpose devices on which we can run anything we want. We still have that with some computers, but we don't with our mobile devices. There, with no meaningful exceptions, we are still vassals of Apple, Google and Microsoft.

Fortunately, Android is a breed of Linux. In the Matrix of mobile, Linux is Zion.

Wake up, Neo. ■

Doc Searls is Senior Editor of *Linux Journal*. He is also a fellow with the Berkman Center for Internet and Society at Harvard University and the Center for Information Technology and Society at UC Santa Barbara.

|||||

Send comments or feedback via
<http://www.linuxjournal.com/contact>
or to ljeditor@linuxjournal.com.